

Grado en Ingeniería en Tecnologías de Telecomunicación  
2018-2019

*Trabajo Fin de Grado*

# AN IN-DEPTH REVIEW OF SOMs WITH APPLICATIONS

---

Óscar Manuel Jiménez Rama

Tutor

Eduardo García Portugués

Leganés, septiembre 2019



*[Incluir en el caso del interés en su publicación en el archivo abierto]*

Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**



## ABSTRACT

The development of internet has opened a new gate from where data can be collected. It is expected that in 2020, each human being will create 1,7 MB of information per second. The management of this huge quantity of data brings new problems and limitations. New analysis techniques are needed to overcome such obstacles to ultimately extract actionable insights from high volumes of data. The adequate analysis of this massive information, not only can be used in the business world to solve problems and take optimal decisions, it also pushes forward research and innovation in many fields of study.

From a statistical point of view, more information not always leads to better predictions: when data “grows”, this usually implies the analysis of higher dimensional spaces in which more complex data structures may be present. Dimensionality reduction techniques, such as Principal Components Analysis (PCA), were conceived to give solution to this problem. Many of these techniques are of a linear nature and, as a consequence, hold limitations when applied to complex data structures. For this reason, many non linear procedures have been proposed in the last decades. Self-Organizing Maps (SOMs) have been gaining popularity over the years with the rise of computing power and since they bring a distinctive approach for high dimensional data inspection focusing on data clustering. SOMs ability to map high dimensional data into a low-dimensional space (commonly two-dimensional) is motivated by certain evidences found in cerebral nature. More specifically, with the studies regarding auto associative memory and its relation with neurons ability to learn.

In this Bachelor Thesis, a complete review of SOMs is carried out. First, theoretical foundations are addressed, contextualizing the technique in its corresponding category within machine learning. Second, an in-depth description from a mathematical point of view is presented with a pedagogic aim. Third, software implementation is discussed using practical examples applied to synthetic data, in order to illustrate how the algorithm works in practice. Finally, an application of SOMs, related with the performance of football players, is presented in order to exploit its properties and extract insights from data.

The codes of the examples can be openly accessed from the following GitHub repository:

<https://github.com/Oscarm96/SOMexamples>

### **Keywords:**

Self-Organizing Maps, Data Science, Unsupervised Learning, Dimensionality Reduction.





## **AGRADECIMIENTOS/ACKNOWLEDGMENTS**

En primer lugar debo agradecer la ayuda de mi tutor Eduardo García Portugués. Este trabajo no hubiera sido posible sin su orientación.

A mi familia por el apoyo que me han regalado desde que era pequeño y la dedicación para seguir creciendo en todos los ámbitos, no solo los académicos.

Finalmente a todos los profesores de la Universidad Carlos III de Madrid, que me han formado como ingeniero y me han alentado a buscar mayores aspiraciones.



## CONTENTS

1. INTRODUCTION. . . . .	1
1.1. What is a <i>Self-Organizing Map</i> ? . . . .	1
1.2. The context for SOMs . . . . .	2
1.2.1. Relation with ANNs . . . . .	2
1.2.2. Dimension reduction techniques . . . . .	3
1.3. Project objectives and contributions . . . . .	4
1.4. Regulatory framework . . . . .	5
1.5. Project budget . . . . .	5
2. FOUNDATION OF SELF-ORGANIZING MAPS . . . . .	7
2.1. Historical development and biological similarities . . . . .	7
2.2. Mathematical formulation . . . . .	7
2.2.1. Inputs, outputs and basic networks architecture . . . . .	8
2.3. Learning process . . . . .	12
2.3.1. Initialization . . . . .	12
2.3.2. Competition. . . . .	13
2.3.3. Cooperation. . . . .	13
2.3.4. Adaptation . . . . .	15
2.4. Supervised SOM . . . . .	16
2.5. Practical demonstration of algorithm. . . . .	16
2.6. Metrics of quality and <i>U</i> -matrix . . . . .	21
2.6.1. Quantization error . . . . .	21
2.6.2. Topographic error . . . . .	21
2.6.3. Trustworthiness and neighborhood preservation . . . . .	22
2.6.4. <i>U</i> -Matrix . . . . .	22
3. GRAPHICAL VISUALIZATION & NUMERICAL ILLUSTRATIONS . . . . .	24
3.1. Software. . . . .	24
3.2. Visualization types . . . . .	24

3.3. Illustrations . . . . .	25
3.3.1. Gaussian case. . . . .	26
3.3.2. Mixture of Gaussians . . . . .	29
3.3.3. Non linear patterns . . . . .	32
4. CASE STUDY . . . . .	36
4.1. FIFA 19 complete player dataset . . . . .	36
4.2. Classification of soccer players with respect to their field positions: part 1. . .	37
4.3. Classification of soccer players with respect to their field positions: part 2. . .	42
4.4. Case study conclusion . . . . .	44
5. CONCLUSIONS . . . . .	46







# 1. INTRODUCTION

Data science, along with big data, has been gaining popularity and interest over the past two decades. Data science can be seen as a superposition of different fields involving math and statistics, computer science and specific knowledge of the area applied to. The goal for a data scientist is to bring applicable insights by means of data analysis in order to, eventually aiding an optimal decision-making. The majority of real life applications bring data scientist an important trade-off situation: choosing between models with high explainability or optimal performance. Usually, opting for higher model complexity reduces the ability to interpret or explain what has been taken into account to produce a certain output. Model explainability becomes especially important in business intelligence or product management applications. Business decisions need to be performed using insights than can be explained in human terms rather than relying in a prediction lacking of interpretability. Here is where SOM (Self-Organizing Maps) finds its biggest virtue: data visualization. Being able to inspect high-dimensional data in a more “human-friendly” dimension (i.e. 2-D or 3-D) automatically facilitates data comprehension.

The incremental use of techniques based on Artificial Neural Networks (ANN) over the last decades, and the limited use of SOMs at the beginnings of its lifespan, caused this procedure to be kept under diminished relevance. As a dimension reduction tool, SOMs has as competitors the classical linear approaches given by PCA (Principal Components Analysis) or MDS (Multi-Dimensional Scaling) that were used as primary option due to, for instance, less computing requirements.

Development of new systems and improvements in information retrieval and filtering, natural language recognition and specific biological studies among others, have returned to throw interest again in this forgotten approach with origin in the early '80s. This is usually the historical trend that ANN algorithms suffered. In absence of sufficient technology and computing power for their usage, other linear techniques that consumed less resources were selected, despite the fact that the mathematical foundations already existed. Additionally, “resurgence” of ANN with deep learning could bring future for SOM improvements.

## 1.1. What is a *Self-Organizing Map*?

The main objective behind SOM (Self-Organizing Map) algorithm is mapping similar patterns of data from an input space to contiguous discrete points in the output lattice. Therefore, classes are not explicitly defined before the learning process, acquiring the category of unsupervised<sup>1</sup> learning algorithm. In general terms, SOM is treated as a clus-

---

<sup>1</sup>There exist SOM variants implementations that support supervised learning (Kohonen et al., 2001, page 215), using labels in the learning process. Used for classification goals.



tering algorithm rather than a classification technique, despite the existence of supervised implementations that are mostly used for classification problems.

SOM makes use of an ANN which represents the output layer and is considered a competitive learning algorithm as opposed to other learning processes based on gradually error reduction to optimize a cost function using back-propagation. This means that each point (node or neuron) located at the output space or lattice, contend to react or be stimulated by a subset of input data. The effect of this competition yields a process in time in which lattice nodes representatives (weight vectors) move towards located areas in the input space, where points affected by similar characteristics from input data, become closer and thus, organized in space. Distance proximity represents statistical similarity. The further a node is from another, the less similar they are.

SOM's main advantage is the ability to simplify high-dimensional data problems and present the results in an easy interpretable way. Reduction in dimensionality plays a key role for making results understandable for humans. One important downside of SOM algorithm is its performance when applied to small data sets or those containing redundant features. Results in these conditions may lead to inaccurate or unclear interpretations.

With a suitable problem, SOMs can bring many applications with high diversity in numerous fields. SOMs have been applied in economics (Resta, 2012), industrial processes (Benítez-Pérez and Benítez-Pérez, 2010), demographics and sociology (Helsinki University, 1992; Huijane and Flavius, 2012), biology (Saevanee et al., 2012; Zhang and Fang, 2012), language recognition and text mining (Kohonen et al., 2000) and meteorology (Angeli et al., 2012).

## **1.2. The context for SOMs**

In this section, SOMs will be placed in the correspondent category within the global perspective of machine learning.

### **1.2.1. Relation with ANNs**

As mentioned before, input data for SOM has not been categorized (there are no labels attached to the data), allocating it as unsupervised learning using ANNs. Models trained using supervised learning, base their classification decision of new data by extracting the absence of similarity features, properly identified in the training phase by means of labels. As opposed to labeled data, non-categorized learning techniques search for patterns between raw data, establishing relationships within the set.

Under the context of neural networks and starting from a general point of view, unsupervised learning is a type of self-organizing *Hebbian learning* (Hebb, 1949). By inspecting the previous name we can distinguish two parts: self-organized and Hebbian learning. Firstly and widely speaking we refer to self-organization as a process where elements

(data) of a disordered system converge to a global ordering as a result of local interaction between subsets within the system. Secondly, *Hebbian theory* is an attempt to explain the associative learning process of the brain neurons, also called *synaptic plasticity*. Hebb's basic principle is usually summarized in: "Cells that fire together wire together". Regarding the short form of Hebb's rule, it comes to explain how the interaction of neurons, somehow related by proximity and function, is crucial for learning.

Artificial Neural Networks (ANN) architectures were created emulating Hebb's basic rule that ultimately translates in the way their weights update. However, not every unsupervised learning technique uses the same update rule to "extract" raw patterns from data. Usually, ANNs are composed by different layers. The activation of one unit in an earlier layer causes a chain reaction that will affect several units from latter layers. Additionally, these units can be performing local non-linear operations to optimize a cost function than can be seen as a measure of how far we are from an optimal ordered state. This direction can also be inverted, performing a chain reaction from latter layers to earlier ones. This is often the case of back-propagation to perform partial derivatives and compute the gradient of the cost function.

SOMs differ from usual ANN techniques, in its characteristic competitive learning implementation. Each time an input observation is shown to the algorithm, a winner unit from the output layer will be selected, based on its similarity (usually using the Euclidean distance) to the datum shown. The winner unit activated by the input observation will affect contiguous nodes in the same plane rather than subsequent layers. The intensity of which a neuron affects others is determined by means of a neighborhood function which uses distance as the influence factor. The closer a neuron is to the unit that has been fired, the greater the weight update will be.

### **1.2.2. Dimension reduction techniques**

Dimensionality reduction techniques were conceived to bring a solution to the *curse of dimensionality* (Bellman, 2015). Such problem appears as the dimension of the space in which the data lives grows, resulting an increment of the "space vacuum" that requires from more observations to be filled in. Dimension reduction methods can be categorized in two classes depending on the approach to achieve reduction of high dimension datasets: *feature selection* or *feature extraction*.

Feature selection aims to select a subset of variables in the original data set to increase performance for multiple goals regarding data analysis. On the other hand, feature extraction or projection, transforms the original set of variables to a lower dimensional space to ease manipulation or improve predictions or classifications. Dimension reduction methods can be linear or non linear. PCA (see, e.g., (Hastie et al., 2001, page 547)) is the most popular linear procedure. PCA generates a new space along the directions of maximum variability between data points, also called, principal components. It does so by linear transformations of the original space.

SOM can be thought as a “non linear” PCA (Wehrens, 2011, page 67), with one main difference: SOM preserves the topological structure of data from the input to the output space in a more accurate manner due to its non linear behavior, allowing SOM a better fit for complex data structures or distributions. Self Organizing Maps performance in non linear distributions is significantly more effectively than in PCA (Figure 1.1, retrieved from <https://commons.wikimedia.org/wiki/File:SOMsPCA.PNG>).

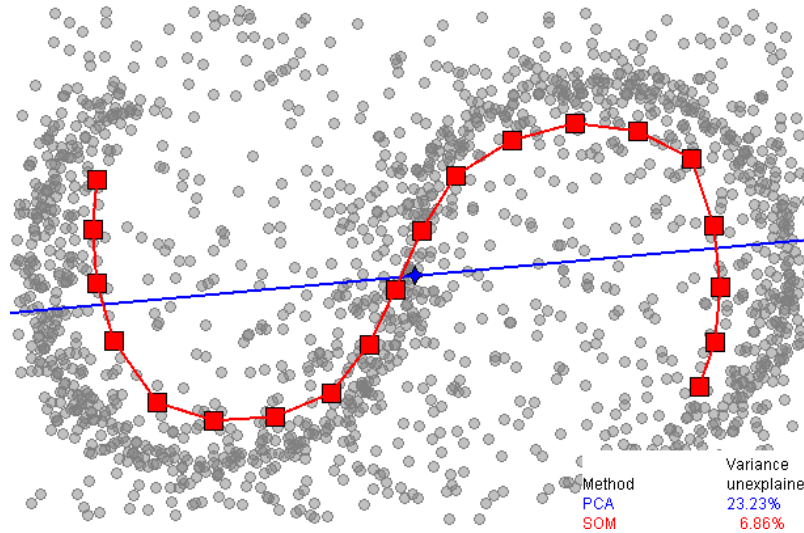


Fig. 1.1. SOM and PCA applied to a non-linear distribution. The red line represents the SOM output (20 nodes) with a 6.86% of variable unexplained, opposite to PCA’s blue line with a 23.23%.

PCA usually projects all the points around the center, making the visualization of local structures difficult if they are far from the center. In simple words, PCA is not a good choice for non linear data. SOM, on the other hand, overcomes PCA in the aforementioned problem.

### 1.3. Project objectives and contributions

This document is organized as follows.

First, Chapter 1 is used as an introduction to place SOM in its corresponding category within machine learning. Moreover, introduction brings a general overview of the basics behind the studied technique.

Chapter 2 gives a deep explanation of the algorithm. Prior to the mathematical scope, an historical context (Section 2.1) of its birth along with the prior biological investigations that contributed to the main idea will be explained. During the mathematical formulation (Section 2.2) a generalization of the initial configuration regarding data and architecture is addressed. Learning process (Section 2.3) will be covered next. Algorithm steps and their different variations will be explained. Additionally the most important tuning parameters will be discussed giving graphical support to comment their effect and relevance. In

order to close the first part of the formal explanation a simple example (Section 2.5) will be used to see a practical demonstration applied to real synthetic data, where the most relevant conclusion seen in previous sections will be proven. Finally a set of quality metrics (Section 2.6) are given for later use.

Chapter 3 comprises the set of tools that we can currently use to apply SOM (Sections 3.1 and 3.2), besides using them to illustrate (Section 3.3) the behavior applied to a set of particular well known cases of synthetic data.

Chapter 4 is dedicated to draw insights and analyze data from a bigger data set taking advantage of the SOMs visualization properties. The standard procedure for inspecting data will be the selected order: data base explanation, data formatting, selection of tuning parameters and conclusions.

## **1.4. Regulatory framework**

The main programming language used to create the examples was R (R Core Team, 2019), an open source code that is distributed as a GNU package. Such package is legally regulated under a General Public License (GPL). GPL allows users to copy, modify or distribute the software, protecting the authors of new implementations using R language, as regulated in the Intellectual Property Law (Intellectual Property no 28, 1995).

The Integrated Development Environment (IDE) used for developing with R was RStudio (RStudio Team, 2019). The software is protected under the Affero GNU GPL v3 that gives the right to use the software freely.

Regarding the tool to create this document, TeXStudio (TexStudio, 2019), powered by L<sup>A</sup>T<sub>E</sub>X (LaTeX, 1984) has been used. The first has a GPL license and the last is protected under L<sup>A</sup>T<sub>E</sub>X Project Public License (LPPL).

## **1.5. Project budget**

This section summarizes the present Bachelor Thesis cost. The results of this analysis is by any means exact and only represents an estimate.

Project budget will be broken down into three categories: software, hardware and labor costs. Cost computations do not take into account economical value depreciation.

Due to the pedagogic aim of this work, the resources used were mainly free regarding software. Firstly, the writing tool used to create this document has been TeXStudio for Mac operating system, powered by L<sup>A</sup>T<sub>E</sub>X. Its a free software package. MATLAB distribution has been used to create graphical illustrations in Section 2.5. Its license is free due to the Campus-Wide License that Universidad Carlos III de Madrid holds. RStudio was the main IDE used for creating numerical and graphical applications of the algorithm under study. It is as well a free distribution package. Finally, Grafio 3, a diagramming appli-

cation for iOS, used for creating theoretical illustrations, had a cost of 11€. Therefore, the software cost is:

$$\text{Cost}_{\text{SOFTWARE}} = \text{TeXStudio} + \text{MATLAB} + \text{RStudio} + \text{Grafió} = 0 + 0 + 0 + 11 = 11\text{€}.$$

Hardware cost only consists in a single laptop (MacBook Pro from late 2018), acquired using students discount at a price of 2.025€. Thus:

$$\text{Cost}_{\text{HARDWARE}} = \text{Laptop} = 2.025\text{€}.$$

In order to estimate the labor cost dedicated to the current project, the minimum salary ranges associated to engineers, summarized in *Disposición 542 del BOE núm. 15 de 2017* (Disposición 542 del BOE núm. 15 , 2017) <sup>2</sup> and shown in Figure 1.2, will be taken into account.

	Mes × 14	Anual
Nivel 1. Licenciados y titulados 2.º y 3.º ciclo universitario y Analista . . . . .	1.687,02	23.618,28
Nivel 2. Diplomados y titulados 1.º ciclo universitario. Jefe Superior . . . . .	1.253,16	17.544,24
Nivel 3. Técnico de cálculo o diseño, Jefe de 1.ª y Programador de ordenador . . . . .	1.208,40	16.917,60
Nivel 4. Delineante-Proyectista, Jefe de 2.ª y Programador de maq. Auxiliares . . . . .	1.107,87	15.510,18
Nivel 5. Delineante, Técnico de 1.ª, Oficial 1.ª Admtvo. y Operador de ordenador . . . . .	968,23	13.555,22
Nivel 6. Dibujante, Técnico de 2.ª, Oficial 2.ª Admtvo., Perforista, Grabador y Conserje . . .	834,17	11.678,38
Nivel 7. Telefonista-Recepcionista, Oficial 1.ª oficios varios, y Vigilante . . . . .	806,20	11.286,80
Nivel 8. Auxiliar Técnico, Auxiliar Admtvo., Telefonista, Ordenanza, Personal de limpieza y Oficial 2.ª oficios varios . . . . .	750,38	10.505,32
Nivel 9. Ayudante oficios varios . . . . .	698,24	9.775,36

Fig. 1.2. Table of salary levels retrieved from *Disposición 542 del BOE núm. 15 de 2017*.

The base salary corresponding to graduated engineers (2nd level) of 17.544, 24€ annually or equivalently 9, 75€ per hour has been selected for the computation of the labor cost. Estimating a total volume of work hours of 350 hours, the total cost regarding labor is:

$$\text{Cost}_{\text{LABOR}} = 9,75 \frac{\text{€}}{\text{hour}} \times 350 \text{ hour} = 3.412,5\text{€}.$$

Adding the three cost divisions obtained previously, a total budget of:

$$\text{Cost}_{\text{TOTAL}} = \text{Cost}_{\text{SOFTWARE}} + \text{Cost}_{\text{HARDWARE}} + \text{Cost}_{\text{LABOR}} = 11 + 2.025 + 3.412,5 = 5.448,5\text{€}.$$

<sup>2</sup>Retrieved from <https://www.boe.es/boe/dias/2017/01/18/pdfs/BOE-A-2017-542.pdf>.

## **2. FOUNDATION OF SELF-ORGANIZING MAPS**

### **2.1. Historical development and biological similarities**

Self-Organizing Maps were firstly formalized and introduced by the Finnish professor Teuvo Kohonen in early 1981, although the core idea of the technique was already conceived in 1976. The study of models for auto-associative memory was the origin for the creation of SOM.

Studies trying to respond the question of how brain learns and structure neurons to make them react upon external inputs captured by sensorial organs, have found that groups of neurons, close to each other, stimulate when receiving certain patterns from the outside. This fact consolidates the existence of neurons topologically arranged in the brain to identify and differentiate between stimuli externally captured.

Additionally, researchers have found that neuron's electric reaction to these patterns decrease as we move away from the center, where this activation is maximum. In other words and giving a statistical context, neurons located at the borders of clusters mapping a particular stimulation, are less relevant in order to identify the input patterns. This is precisely, the primary concept behind SOMs.

In spite of reaching this relevant conclusion, studies still can not assure if these neurons maps are dynamically produced by means of a learning process or are predetermined by genetic information.

### **2.2. Mathematical formulation**

In this section we will describe the global structure of our problem from a mathematical point of view. We will describe the phases of SOMs algorithm, breaking down every aspect of the map formation, the functions that enable learning along with their hyper-parameters and the structure of the most common network topologies used.

### 2.2.1. Inputs, outputs and basic networks architecture

Let be a set of  $N$  vectors or samples (inputs) of  $D$  components (features), conforming a sample in  $\mathbb{R}^D$ . Each  $\mathbf{x}_i$  is an observation in  $\mathbb{R}^D$  :

$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$$

$$\left. \begin{array}{l} \mathbf{x}_1 = (x_{1,1}, x_{1,2}, \dots, x_{1,j}, \dots, x_{1,D}) \\ \vdots \\ \mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,j}, \dots, x_{i,D}) \\ \vdots \\ \mathbf{x}_N = (x_{N,1}, x_{N,2}, \dots, x_{N,j}, \dots, x_{N,D}) \end{array} \right\} \text{N observations}$$

We refer to the  $i$ -th datum of the data as  $\mathbf{x}_i$  and its  $j$ -th component as  $x_{i,j}$ .

Input space produces the input layer, a set of  $D$  nodes representing the input dimension or the number of features belonging to our input data.

The output space or layer consists in a set (1-D) or a grid (2-D or 3-D) of units (neurons) which in practice can be arranged in a coordinate system up to three dimensions, although most common maps are bi-dimensional. The size of the output grid has to be determined prior to training.<sup>3</sup> The number of neurons has to be set in order to avoid having notably more or less nodes (clusters) than patterns in the original data. Figure 2.1 represents the graphical representation of the topology.

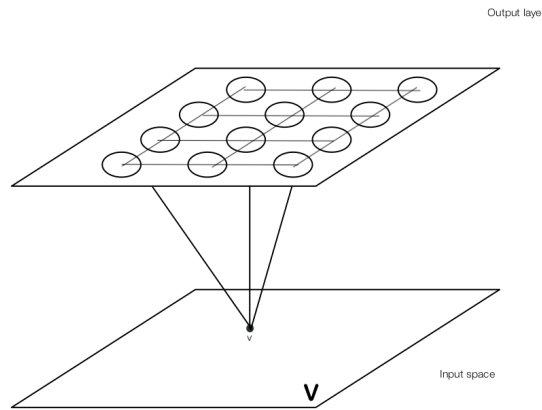


Fig. 2.1. Input space and output layer.

Each circle in the output layer represents a neuron or a node. Each node is connected to their neighbor ones forming a lattice. Lateral connection between units are used to

<sup>3</sup>Preselection of the grid size is one of the disadvantages of SOM. Growing SOM (GSOM) (Bauer and Villmann, 1997) are a variant from the original in which the output lattice shape “grows” based on the input and the convergence of the algorithm.

identify the relative position of nodes surrounding each other. For the purpose of comprehensibility, let's formulate the one dimensional version of the output layer:

$$\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_M)$$

$$\left. \begin{array}{l} \mathbf{y}_1 = (y_{1,1}, y_{1,2}, \dots, y_{1,j}, \dots, y_{1,D}) \\ \vdots \\ \mathbf{y}_i = (y_{i,1}, y_{i,2}, \dots, y_{i,j}, \dots, y_{i,D}) \\ \vdots \\ \mathbf{y}_M = (y_{M,1}, y_{M,2}, \dots, y_{M,j}, \dots, y_{M,D}) \end{array} \right\} \text{M units}$$

In the equation above,  $M$  stands for the number of units in the output lattice, representing classes or clusters our data will be mapped in. Each unit has a coordinate position in the grid which is different from the coordinate vector  $\mathbf{y}_i$ . We use  $\mathbf{r}_i$  to specify the lattice position of the  $i$ -th unit.

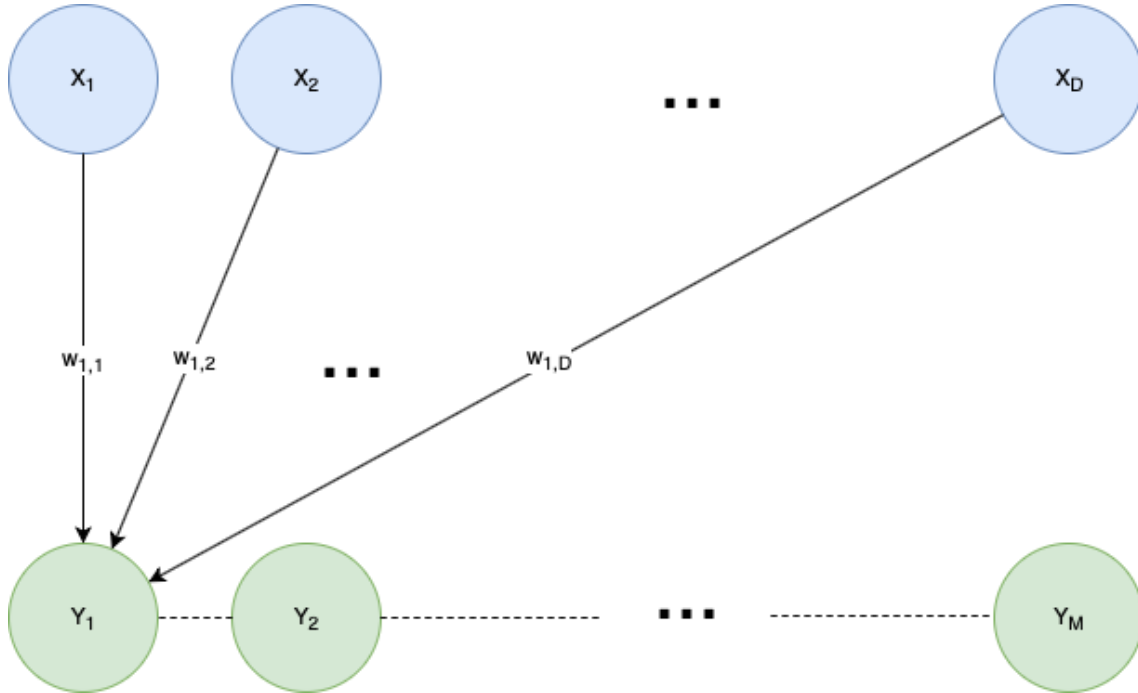


Fig. 2.2. Network topology.

As we see in Figure 2.2, input units are fully connected to every node in the output layer. This special structure distribution receives the name from its creator which happens to be as well the father of SOMs algorithm: Kohonen Network. It is important to mark that, in spite of output lattice units implementing lateral connection, they are not associated to specific weights. Their purpose is to keep track of the surrounding nodes of a given neuron and apply an update to those surrounding the winner unit (latter explained).

Links between layers are represented as vectors of weights. For each output unit exists a  $D$ -dimensional (input space dimension) representation referred to indistinctly as weight



vector or codebook vectors, representing the relation between itself and a subset of input points with similar characteristics (i.e. a cluster). Mathematically:

$$\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m, \dots, \mathbf{w}_M)$$

$$\left. \begin{array}{l} \mathbf{w}_1 = (w_{1,1}, w_{1,2}, \dots, w_{1,j}, \dots, w_{1,D}) \\ \vdots \\ \mathbf{w}_i = (w_{i,1}, w_{i,2}, \dots, w_{i,j}, \dots, w_{i,D}) \\ \vdots \\ \mathbf{w}_M = (w_{M,1}, w_{M,2}, \dots, w_{M,j}, \dots, w_{M,D}) \end{array} \right\} \text{ M weight vectors}$$

Above,  $\mathbf{w}_m$  represents the weight vector of the  $m$ -th neuron, where each component,  $w_{i,j}$ , is the  $j$ -th coordinate in the input space, representing a particular feature. Note that each neuron is associated to a vector with the same dimension of the input layer (i.e.  $D$ ). Therefore, such codebook vectors form a compressed representation of the cluster in the input space. As a concise conclusion, the set of codebook vectors describe the compact form of a group of samples that hold similar features.

Topographic map formation is done in such a way that near neighbor units  $\mathbf{y}_i$  and  $\mathbf{y}_j$ , with  $\mathbf{r}_i$  and  $\mathbf{r}_j$  lattice positions respectively, are contiguous in the output space while weight vectors of such units (i.e.  $\mathbf{w}_i$  and  $\mathbf{w}_j$ ) are not necessarily close to each other in the input space at initialization. Even when the map has been created, weight vectors associated to contiguous units can be distant from each other. As learning progresses and updates are performed, distant weight vectors belonging to contiguous output nodes, will tend to get closer in the input space, creating a topological association in both spaces (topology preservation).

Forward Mapping (**F**) (Figure 2.3) may or not preserve the input topology as not always neighbor weight vectors are contiguous units as a consequence of the non-equivalence between the dimension of input space and output layer. This enables the creation of quality factors to measure topology preservation (see Section 2.6).

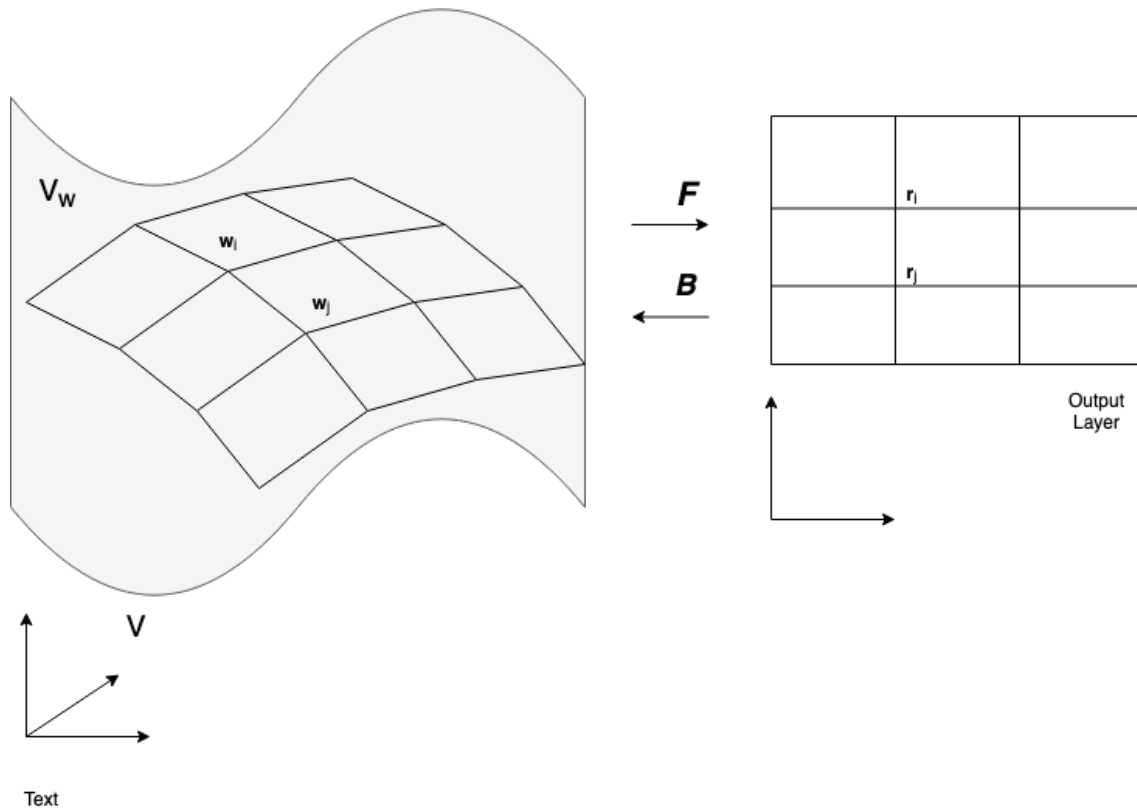


Fig. 2.3. Forward and backward mapping of weight vectors

Connection between input and output units is said to be forward, i.e directed from input to output. SOM's neural arrangements are known as feed-forward networks, as opposed to back-propagation networks for one main reason. Convolutional neural nets perform a two directional iteration, a forward oriented from input to output layer and a contrary one to update weights performing derivatives with the final goal of algorithm convergence. SOM's architecture is designed so that, during learning, forward direction is exclusively used. Back-propagation optimizes a cost function by means of performing partial derivatives in chain. SOM lacks of an cost function to optimize and therefore it is impossible to compute a global error measure.

In spite of SOM not having a global cost function, it does solve an optimization problem when the neighborhood function is not involved in the learning process. Further details will be given in latter sections.

Going back to mathematical formulation, a specific weight coordinate represents the relevance of that feature being mapped to an specific region of the map. SOMs topology preserving property, is possible due to the existence of connections between units in the output layer, allowing the maintenance of a fixed reference along the iterative process. When a particular node is identified as the best matching unit, not only weights associated to itself updates, but also the weight vectors surrounding the winner.

In practice, SOM algorithm is used mainly for clustering. Classification is not the goal but it can be used for such purpose. Output units represent a region in the input

space, where their weight vectors live, so that each area limited by the bisector lines connecting adjacent weight vectors represents a region feature of the input continuous space (Figure 2.4). Therefore, input space is divided in portions creating a Voronoi tessellation, consequence of the minimum Euclidean distance rule.<sup>4</sup>

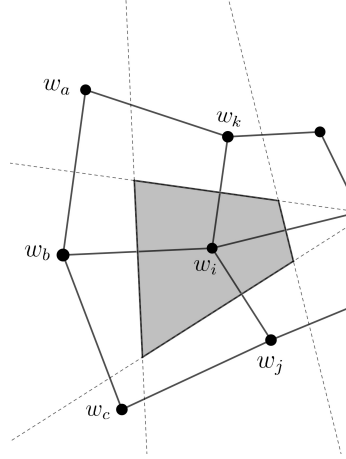


Fig. 2.4. Portion of output layer along with the region, associated with node  $i$ , representing a feature in the continuous input space.

## 2.3. Learning process

In every SOM's variant, from the basic one to the most complex, we can distinguish four main phases in the self-organization process: (1) initialization (Subsection 2.3.1); (2) competition (Subsection 2.3.2); (3) cooperation (Subsection 2.3.3); (4) adaptation (Subsection 2.3.4).

Due to the lack of general consensus among literature to create a set of default phases in the algorithm process, we have opted for the separation that allows more detailed explanation.

### 2.3.1. Initialization

Initialization is the first phase of SOM algorithm. It consists in initializing weight values of each unit in the output lattice. Performance and learning quality of the algorithm are determined by this phase. A poor or deficient initialization could not lead to a successful convergence of SOM.

There are two principal approaches of setting the initial weight values:

- Random Initialization (RI).

---

<sup>4</sup>Voronoi regions to describe clusters are also another difference with other clustering techniques, as opposed to radial geometrical forms (kernels).

- Principal Components Initialization (PCI).

RI uses random values from the sample data to set initial variables. On the opposite, PCI initializes weights from the subset spanned by a fixed number of principal components. SOMs time performance speeds up using PCI due to the pre-ordering of PCA. According to Akinduko et al. (2016), a study comparing both approaches, used a method criterion to evaluate performance and learning based on the fraction of the variance unexplained. The results could not bring conclusions in relation to PCI, but showed that RI accomplished better performance with non-linear against quasi-linear distributions.

### 2.3.2. Competition

After initialization of weight vectors, learning starts. Competition stage identifies the *best matching unit* (BMU) in the discrete output space for a particular vector sample, i.e. the most similar neuron, denoted as  $\mathbf{y}_m^*$ , with respect to the given input. First we need to define a discriminant function to measure similarity between units. Euclidean distance is the most popular although dot product can be used (Kohonen et al., 2001, page 115).<sup>5</sup>

For each input sample  $\mathbf{x}_n$  with  $n = 1, 2, \dots, N$  (depending on the time step), we determine the BMU or the “winner” of the contest:

$$\mathbf{y}_m^* = \arg \min_m \| \mathbf{w}_m - \mathbf{x}_n(t) \|$$

The “winner” neuron has, among the rest, the closest weight vector with respect to the input data at a given time step  $t$ . Cooperation between units around is carried out at the next phase.

### 2.3.3. Cooperation

The first condition to create a topological map is that weight updates are not narrowed to the BMU but also to the ones that hold some grade of similarity with the sample data responsible of firing the winner neuron. During the first iterations of the algorithm, neurons topologically close to each other are not statistically similar, however with time, topological distance and statistical similarity will tend to converge into the same concept.

This phase selects which nodes surrounding the BMU are going to be sensible for weight updates. As occurs in the human brain, the peak of the electric excitation is produced in the center of a particular brain portion. As we move towards the edges of the area stimulated by one particular pattern or input, neurons experiment a weaker stimulation until we reach the border ones where the electrical pulse is minimum. SOMs algorithm tries to adapt this behavior into a bigger weight variation as we move towards the best matching unit (i.e. the center).

---

<sup>5</sup>If data is scaled so that its norm is 1, both metrics are equivalent:  $\|\mathbf{x} - \mathbf{y}\|^2 = 2(1 - \mathbf{x}'\mathbf{y})$ .

We define a scalar function denoted as the neighborhood function represented as:

$$N_{m^*}(m^*, m, t) = \exp\left(-\frac{\|r_{m^*} - r_m\|^2}{2\sigma_N^2(t)}\right), \quad (2.1)$$

where  $\mathbf{r}_{m^*}$  and  $\mathbf{r}_m$  indicate the grid coordinates of units  $\mathbf{y}_{m^*}$  (BMU) and  $\mathbf{y}_m$  respectively. Note that function (2.1) holds a Gaussian shape with standard deviation  $\sigma_N(t)$ . Variance in a Normal distribution sets the width of the curve, hence it determines the reach of the curve together with the number of units affected by the activation of the BMU. Another aspect to mention in relation to variance (i.e.  $\sigma_N^2(t)$ ) is that varies with time steps or iterations. As the iterative process converges, standard deviation decreases according to:

$$\sigma_N(t) = \sigma_{N_0} \exp\left(-2\sigma_{N_0} \frac{t}{t_{max}}\right), \quad (2.2)$$

where  $t_{max}$  is the maximum number of iterations or time step. Decreasing behavior suggests that interaction on the cooperative phase between lattice nodes is more important when disorder is maximum (earlier iterations) yet not powerful when convergence is proximal. Same insight is reflected in Figure 2.5.

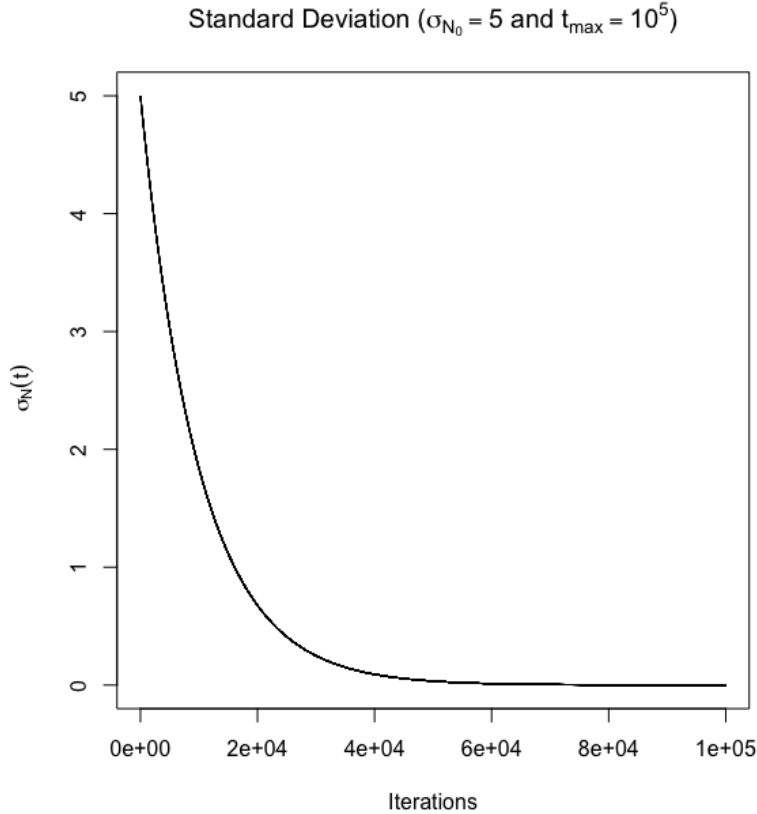


Fig. 2.5. Standard deviation vs number of iterations.

It was said the SOM does not optimize a global cost function. The main explanation is the complexity that such optimization would have. Principal reason is that updates depend on the data point that is shown at a particular time step. This impacts directly on

the weight vectors affected by the update at each iteration, forcing the existence of local cost functions for each output unit that can only be fixed during such iteration. A different BMU together with its neighbors will be selected in the next time step, eliminating the effect of the optimization occurred in the previous iteration. However, when SOMs learning process is close to end, updates are only performed in a single unit, the BMU, or in other words, when neighborhood function does not influence other neurons. In such scenario, it is possible to consider an energy function due to the elimination of a notorious portion of complexity. Effectively, in such circumstances, SOM algorithm optimizes a function to minimize the error between the compact representation of clusters (i.e. weight vectors) and the input data.

### 2.3.4. Adaptation

Here is where actual learning takes place. The update rule changes based on the learning mode.

#### Incremental mode

When incremental mode is used, weights are updated at each iteration, in other words, each time a sample vector is shown. The weight update rule for the  $m$ -th neuron is:

$$\mathbf{w}_m(t+1) = \mathbf{w}_m(t) + \eta(t)N_{m^*}(t)[\mathbf{x}(t) - \mathbf{w}_m(t)],$$

with  $\eta(t)$  being the learning rate as function of time in a monotonically decreasing as seen with the standard deviation, can be implemented in a linear way.

#### Batch mode

Batch mode not only affects the adaptation phase, competition is modified so that for each training point  $\mathbf{x}_n$ , a BMU is selected. As the name (batch) indicates, for each iteration, the whole set of samples  $\mathbf{X}$  is shown, assigning a winner neuron for each observation before updating weights. In this way, each iteration produces as many winners as the sample size. Once the entire batch is processed, the following weight update rule is applied to each output neuron in the lattice:

$$\mathbf{w}_m(t+1) = \frac{\sum_{n=1}^K N(m, m^*, t) \mathbf{x}_n(t)}{\sum_{n=1}^K N(m, m^*, t)}. \quad (2.3)$$

Batch mode may permit that one single unit is the winner for numerous training points. We denote  $K$  to be the number of samples for which the unit  $y_{m^*}$  is their BMU. According to equation (2.3), larger updates will take place for units where  $K$  is larger. In other words, the neuron which represents a larger number of samples, affect more units close to itself. Recall that the neighborhood function is taking the  $m$  unit, but the winner neuron for a particular sample  $\mathbf{x}_n$  (i.e.  $m^*$ ) changes from  $n = 1, \dots, K$ , along with the corresponding input sample.

## 2.4. Supervised SOM

In spite of the fact that SOM was firstly introduced as an unsupervised technique, there exists numerous implementations supporting a supervised version. In fact, it is the most commonly used variation of SOM when applied to real datasets. Supervised SOMs, as one can imagine, incorporates sample's true labels during training. This is done by adding an extra parameter of information to the input data, keeping different distance scales between labels and features. Data components and labels are kept in different layers of the map. Both layers learn independently and are combined at the end to obtain the final map.

Not only it is possible to obtain better predictions than using hierarchical clustering<sup>6</sup> after performing an unsupervised SOM, it also facilitates the map interpretability and enhances stability as well. It is also possible to assign a numerical weight value between 0 and 1 to specify how much effect the labels influence in the final map with respect to data features. By defect the influence is distributed in half for labels and features.

A practical demonstration is shown in Chapters 3 and 4.

## 2.5. Practical demonstration of algorithm

Let us show a practical example of SOM's performance<sup>7</sup>, giving a detailed description of the interpretability as the algorithm converges.

It has been trained a  $5 \times 5$  rectangular grid (output layer) using sequential SOM algorithm over a small dataset ( $N = 25$ ) in order to facilitate understanding with a simpler situation.

Data is generated from a bi-dimensional normal distribution with zero mean and identity covariance matrix:

$$X \sim \mathcal{N}([0, 0], \mathbf{I}_2)$$

The next figure shows the position of data samples (red crosses) in the bi-dimensional plane along with the codebooks initial coordinates of each unit in the output layer (black circles) (2.6). Connections between weight vectors can be seen as an auxiliary information to keep which codes are associated to contiguous units in the output lattice. As said before, neighbor weight vectors around the winner codebook will perform greater updates.

---

<sup>6</sup>Documentation for Kohonen library (see Section 3.1) explains how to use hierarchical clustering after an unsupervised SOM map.

<sup>7</sup>The software used can be seen in Section 3.1

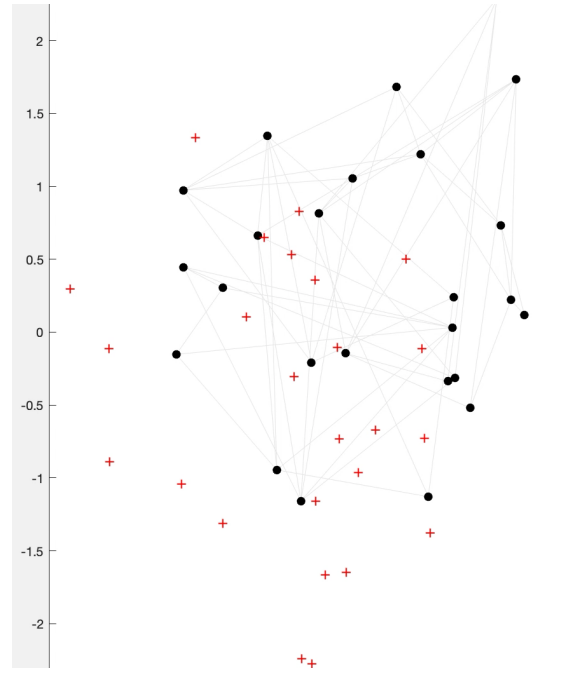
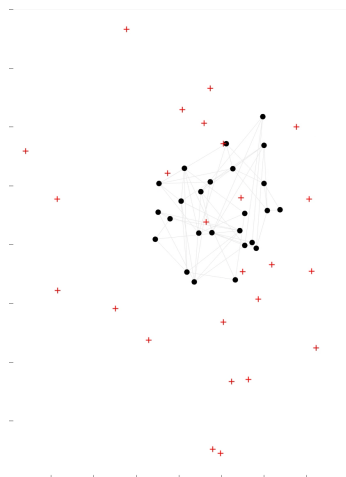


Fig. 2.6. Initial codebook position

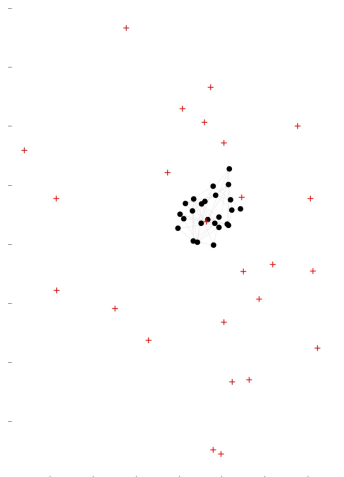
As we mentioned in the previous section, weight vectors have to be initialized to begin training process. Random initialization has been used for this example. Each codebook vector represents a data subset of the input space holding similar characteristics. At first, as we notice in figure (2.6), initial codebooks are placed without any criteria based on input data (randomly). In fact, we can see that black circles are somewhat displaced to right with respect to the input data cloud. Also note that connected black circles (belonging to contiguous output units) can be placed in opposite corners of the plane.

Intuitively, one can think that the algorithm may take more time to converge, due to the distance between further codebooks and data points. If random initialization locates weight vectors such that the mean distances between these and data points is smaller, SOM incremental learning will not perform as much updates to fit the data due to partial pre-ordering. This insight shows how initialization is important when it comes to time efficiency. Moreover, poor initialization can lead to bad results or a non-convergence scenario, especially when small data sets are processed.

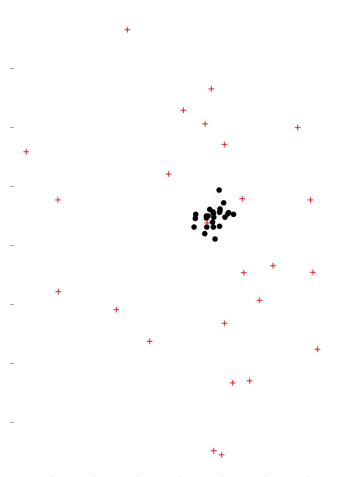




(a) 10/300



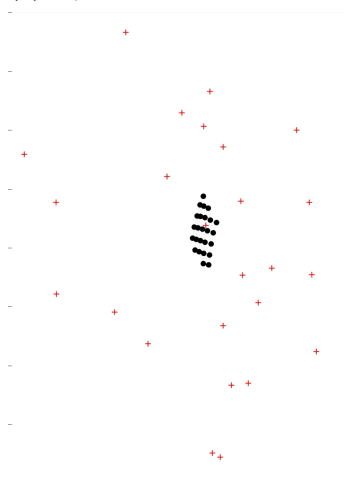
(b) 20/300



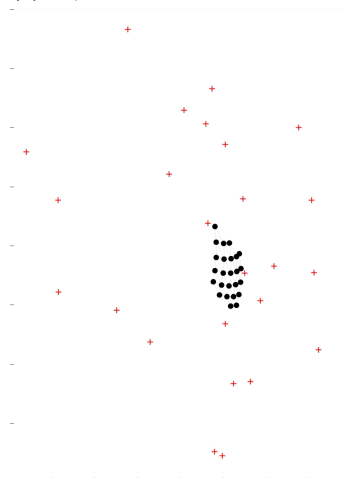
(c) 30/300



(d) 60/300



(e) 80/300



(f) 120/300

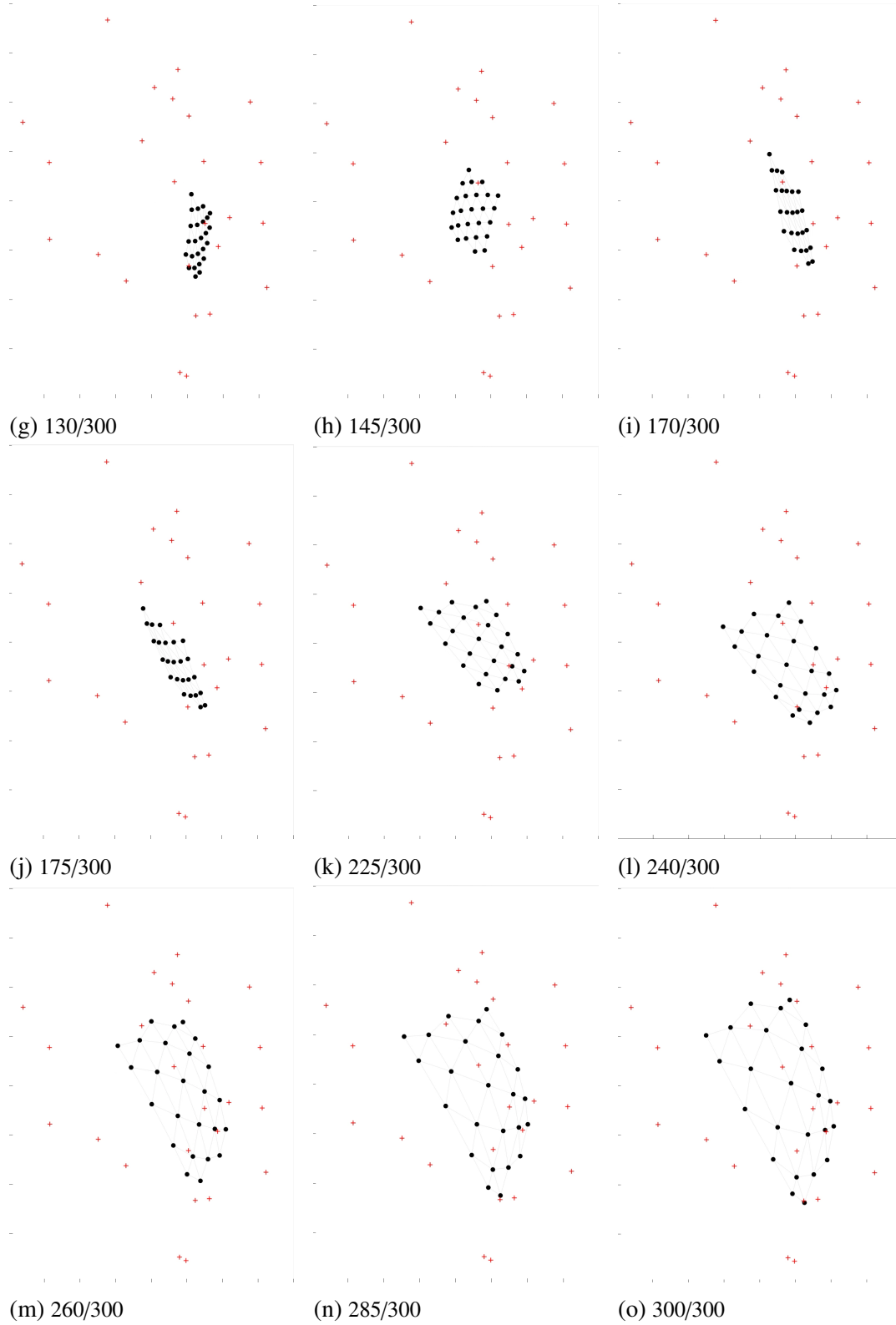


Fig. 2.7. Evolution over time steps.

It is known that map creation goes through three (non-formal) different phases while learning: shrinking, ordering and dilatation. We describe them next:

**Shrinking:** At first, weight vectors are distributed within the input plane lacking any way of organization. Once the first BMUs are excited, distant contiguous weight vectors will tend to shrink towards the middle where the majority of data points are placed. Shrinking phase can be seen in our example in the first row of images in Figure 2.7 (iterations from 0 to 30).

**Ordering:** Due to proportional updates with respect to distance and considering the fact that there exists a natural reference system created in the output lattice, weight vectors become organized in the input plane emulating their geometrical relative form of their original output space. This grid formation being adapted by codebooks can be seen in the second and third row of images in Figure 2.7 (iterations from 60 to 175). Note that, despite distinguishing a grid distribution, there exists a deformation or stretching of such grid produced by the position of input data points. SOM is always fitting the map towards input data and every update is done as an output unit reaction triggered by a subset of input points.<sup>8</sup>

**Dilatation:** Lastly, as a consequence of shrinking, codebook vectors are concentrated in a small region of the input space, becoming far from the surrounded data points. As a counter reaction to this concentration, weights expand in order to fit the data, once a partial organization has been achieved as a result of ordering phase. This is seen in the fourth and fifth row of Figure 2.7, corresponding to iterations from 225 to 300.

Explanation of these three different phases can be described as a result of the decreasing behavior of both the learning rate and the standard deviation (2.2) from the neighborhood function (2.1). Standard deviation influences the “radio” or range of neurons that are affected when the BMU is activated. In the shrinking phase, steps are notable even if weight vectors lay distant from each other, when the width of neighborhood function is broad. Same occurs with learning rate. As the algorithm progresses, the effect of these parameters weakens. Ordering and dilatation phases need more iterations to perform similar displacements to those happening in the shrinking phase.

This brings an important aspect: convergence is highly determined by the decreasing rate of these parameters. If decline is quick, weight vectors may not have enough steps to fulfill the first partial ordering performed in the second phase. In the other hand, in the case that decreasing rate is small, the algorithm may not find a good fit for the data before the maximum number of iterations in the algorithm.

---

<sup>8</sup>Here we can set a strong similitude of animal sensorial neurons excited by an external output (Section 2.1).

## 2.6. Metrics of quality and $U$ -matrix

SOM is considered a Vector Quantization method. Codebook vectors are precisely the “compressed” distribution of the set of samples that represent. It is sensible to define a quantization error in order to measure the loss of information produced by the compression.

SOM’s algorithm does not minimize a global cost function. This only happens when the neighborhood factor has no effect (i.e.  $N(t) = 0$ ), usually at the last iterations. (Kohonen et al., 2001, page 146) explains, based on the previous, the *impossibility of formalizing SOMs algorithm as an optimization problem to find the minimum of the global error quantization*. The core of this unfeasibility resides in the unsolved theoretical problem, involving terms that vary with time (neighborhood function). Although there is not a solution, an approximation exists, the Robbins–Monroe stochastic approximation.

The main two properties of interest to evaluate for SOM are vector quantization and topology preservation. These properties, as many others in learning algorithms, are related in a tradeoff situation. According to Pözlbauer (2004), metrics are often categorized based on the property that are evaluating, however they can also be used to determine hyper parameters of the network such as the grid size.

Next, we will present some practical metrics that help us determine the learning quality of the map under the scope of SOM’s two principal properties previously mentioned.

### 2.6.1. Quantization error

Quantization error can be seen as the quality that a centroid, in our case codebook vector, represents its associated points. It is obtained by the following expression:

$$\varepsilon_q = \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \mathbf{w}_c\|, \quad (2.4)$$

where  $\mathbf{w}_c$  is the codebook vector of the node in which  $\mathbf{x}_i$  has been clustered.

Equation (2.4) does not take into account topology preservation or global alignment of the map. A way of minimizing this error is to increase the number of nodes in the grid so that weight vectors of nodes are less distant. As a trade-off, vector projection quality is penalized, meaning that compressed distribution of the layer units becomes more inaccurate. Also, equation (2.4) brings an analogy to k-means as it can be seen as the within variance of the cluster whose center is  $\mathbf{w}_c$ .

### 2.6.2. Topographic error

Topographic error is one of the simplest alternatives for ranking topology preservation. The computation of this error is done for the whole dataset. For each sample the first two

BMU are calculated, that is, the two nearest centroids or codebook vectors from the input point. If those nodes, associated to the nearest weight vectors, are not adjacent, then it is considered a topology defect. The sum of all topographic errors divided by the sample size ( $N$ ), is called the global topographic error:

$$\epsilon_t = \frac{1}{N} \sum_{i=1}^N u(x_i),$$

where  $u(\cdot)$  is the step function, with unit value in the case that the first and second BMU associated to  $x_i$  are not adjacent, and zero with contiguous best matching units.

### 2.6.3. Trustworthiness and neighborhood preservation

These quality measures quantify topology preservation adding an important focus on a new perspective with respect to the previous metrics. The key point is to determine how big is the difference between relative positions of data points in input space with respect to the output space. Similarly to topographic ratios calculation, we use a parameter  $k$  to set the number of nearest neighbors referred to data points. To compute trustworthiness, first we select the  $k$ -nearest points in output space and raise a counter that increases when one element in the  $k$ -set is not contiguous in input space. Averaging these errors along the whole set and subtracting it to one, the *trustworthiness* is obtained. Its maximum value is 1, meaning perfect preservation. *Neighborhood preservation* is performed in the same way by swapping input and output space. We can say that both metrics are equal but with opposite direction (OUT-IN vs IN-OUT).

According to Venna and Kaski (2001), there exists problems implicitly incorporated in SOMs implementation. The principal obstacle is found in the output space. Projected data mapped in the output layer live in a discrete space. Unless a transformation is performed, points of the same unit will have zero distances between them in the end space. Many approaches have been done to solve the problem, yet only one has been practical. The method is based on averaging data points with same position (output lattice), accordingly to their coordinates at input space.

Both measures can be computed for many depth levels ( $k$ ), that may be considered dependently for each application.

### 2.6.4. U-Matrix

Although U-Matrix or unified distance matrix is not considered a quality measure, it can be useful to visualize how well is data fitted in the lattice. Rather than a deterministic metric (which it is), is popularly used as a representation device that will be covered in the next section. As a superficial explanation, U-Matrix represents distances between adjacent neurons.

For each node, Euclidean distance mean between weight vector of the present neuron and that of the adjacent ones are computed. Normally the number of neurons used in a near neighborhood depends on the grid. If a rectangular grid is used, eight codebook vectors are used; and in a hexagonal grid, six contiguous nodes.

When a particular node is associated to a remarkable high value in the U-Matrix, we can interpret it as if the current neuron, a compact representation of a distribution belonging to a limited set of samples in the set, differ significantly from the closest neurons compressing the information of other subsets. It is another way of saying that U-Matrix can be used for defining the set of clusters. With a visual expression of this matrix, explainability is best.

As said at the beginning, this method can be used to evaluate grid dimensions. Having low values for the total set of nodes proves the need of more units to fit the data set. An opposite scenery would be having high values, denoting large gaps between codebook vectors, thus showing surplus of nodes.

### 3. GRAPHICAL VISUALIZATION & NUMERICAL ILLUSTRATIONS

This chapter is dedicated to demonstrate the practical use of SOM by means of open source software tools. Showing different synthetic data examples we will present the numerous visualization outputs that software implementations of the algorithm hold. As said before, SOM's first advantage is the capability of showing statistical insights over data in a graphical human-friendly way. These practical demonstrations will reveal most of the key concepts that has been discussed so far.

#### 3.1. Software

There exists multiple implementation of SOMs. It is possible to find libraries coded in Python (SOMPY, Vahid Moosavi (2014)), Matlab (MATSOM, Oy et al. (2000)) and R (Kohonen, Wehrens and Buydens (2007)), along with dedicated software (Living For SOM, Martínez-Martínez et al. (2016)). We will focus on investigating the Kohonen R library.

Some of this software has been already employed in Section 2.5 (Oy et al., 2000). It contains pre-made examples for different data distributions along with a set of functions implementing the algorithm.

We will study in detail in Section 3.2 the outcomes of the Kohonen package for concretion purposes.

#### 3.2. Visualization types

This section introduces the map visualizations that can be computed after the learning process. It is needed a brief introduction of their meaning before the practical demonstrations in order to understand their outcomes. Many of these maps have been created in order to facilitate the user analysis of data. Each map can be seen as a global (graphical) measure to highlight different aspects of the information studied.

Most of these types of map utilize the unit's weight vectors and a color code to compute a map visualization. With a single run of the algorithm it is possible to compute all the map visualizations by using the final state of the output units and their information (codebooks).

The following are the most important map visualizations that can be found in the majority of software implementations. The nomenclature used for this description follows the same one as the Kohonen package (Wehrens and Buydens, 2007) but names can vary

depending on the implementation.

- Changes:** Shows a graph where the samples mean distance to their corresponding closest codebook vectors is represented with respect to time steps during the map creation. The resulting curve should present a decreasing behavior.
- Codebooks:** A rectangular grid with dimension determined by grid size (chosen by the user) representing every unit in the output layer. Each unit contains a pie diagram equally divided into the input space dimension  $D$  and with different radius representing the numeric value of the unit's weight vector feature associated to itself. It is usually used to look at first glance the key differences among the grid units.
- Counts:** The output grid is shown in the same way as the codebooks map. Each unit is filled with a color gradient that represents the number of samples associated to each codebook vector. A more common name for this type of visualization is called heat-map, although other category of heat-maps can be computed for other measures apart from counts.
- U-Matrix:** Also referred to as neighbors distances plot (Subsection 2.6.4). It is a heat-map visualization where the sum of all the distances to all contiguous units are mapped into a color code and shown as a grid of units filled with their corresponding color intensity. The main purpose for this map is to visualize boundaries between clusters. Border cluster units are darker and middle units lighter.
- Mapping:** Same output grid is represented, showing all the mapped samples to their corresponding units. Usually is applied when labeled data is used.
- Properties:** Lets the user visualize in a color code the similarities between objects with respect to the map units under a single variable.
- Quality:** Computes the quantization error (Subsection 2.6.1) of each unit and maps it value to a color code. The darker color intensity (small distances) the unit is filled with, the more representative that unit's codebook vector is.

### 3.3. Illustrations

Hereafter the example cases will be discussed. Mainly each demonstration will exhibit a practical procedure on how to take advantage of the earlier map visualization and how the user can exploit them in order to obtain useful insights.

Cases will be presented in an increasing complexity order. The simpler data distributions will be convenient to show various basic SOM properties. To cover specific details and comparisons with other techniques, a much more tailored demonstration will be carried out.



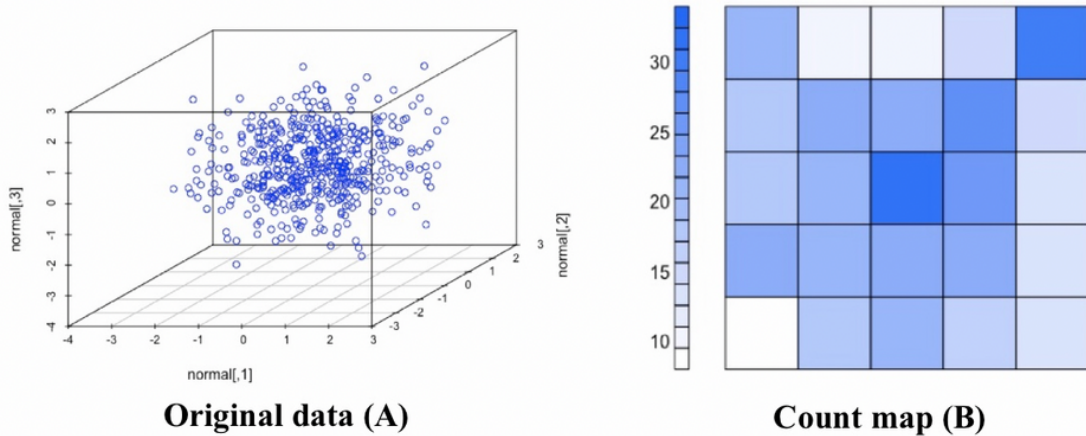
### 3.3.1. Gaussian case

As a dimensional reduction technique, let us show how SOM behaves with a multivariate Gaussian distribution. In the present case, SOM will be applied to a three dimensional Normal distribution.

Before showing any result there are a few remarks that should be mentioned. Firstly, as happens in the totality of cases, when the input dimension grows, the training phase needs more samples to achieve a comparable efficiency as in lower dimension spaces (see the curse of dimensionality in Subsection 1.2.2). Also, topology preservation property will weaken as the dimension grows even if the sample size is sufficient, a consequence of the increasing complexity of mapping  $D$ -dimensional spaces into a (typically) 2-dimensional space. Additionally, prior to training, setting a grid size according to the number of samples is necessary. From this point in advance the recommendation mentioned in Rojas et al. (2015) will be chosen. It sets the *maximum* grid size to be  $5\sqrt{N}$ , where  $N$  is the sample size.

The input data contains  $N = 500$  observations. As said before, data points are generated from a 3-D Normal distribution, centered at the origin and with identity covariance matrix (plot A in Figure 3.1). The SOM model has rectangular grid of size  $5 \times 5$ , within the range mentioned in Rojas et al. (2015), even though grid size does not become an important parameter working at the current volume of the set. Training is performed given the default parameters of the function used, i.e. incremental mode (online) and 100 iterations.<sup>9</sup>

Figure 3.1 shows several visualization outputs, described in Section 3.2:



<sup>9</sup>In this context, iterations are referred to as the number of times the complete set is shown to the input layer, thus the number of time steps are  $500 \times 100$  in this case.

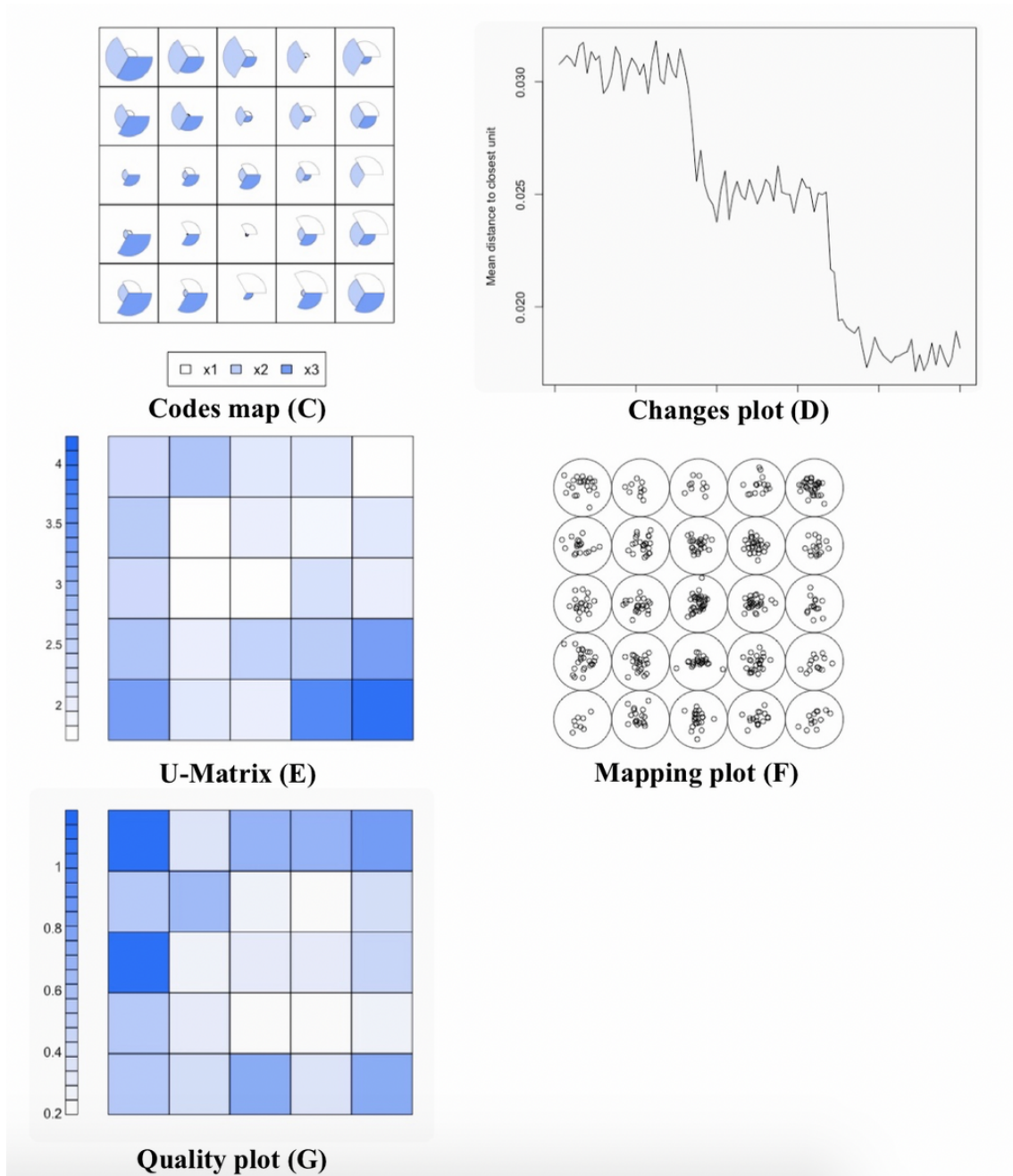


Fig. 3.1. Graph A: Scatter plot of original data; graph B: Count map; graph C: Codebook map; graph D: Changes plot; graph E: *U*-Matrix; graph F: Mapping plot; graph G: Quality plot.

Graph D from Figure 3.1 shows the changes plot. The curve holds a decreasing behavior as expected. As iterations ( $x$ -axis) increase, the mean distance of the samples to their closest codebook ( $y$ -axis) becomes smaller. Furthermore, there exists another detail in the graph that manifests the three different stages of the algorithm: shrinking, ordering and dilatation, mentioned in Section 2.5 and shown in Figure 2.7. Such plot shows three distinguishable levels, each for one stage, where the mean value (mean distance) at each of them decreases as we move to convergence. Note that such decreasing behavior is not monotone, opposed to what would be expected if SOM optimized a cost function.

Count map is shown in graph B from Figure 3.1. It contains the rectangular grid (5x5) where each division represents one unit at the output layer. Darker units contain more samples than lighter ones. Kohonen library numbers the grid units starting at the bottom left corner, increasing as we move towards right hand direction and upward. The unit located at the bottom left is unit number 1 and top right corner unit is set as number 25.

The unit placed at the center is the most populated unit as the dark blue legend shows. More specifically, unit 13 contains 34 samples associated to the unit's weight vector. It can be seen how the counts belonging to the units placed in a near neighborhood from number 13 gradually decrease as their distance increase (blue intensity decreases). This manifests the topology preservation property. In the same way that the majority of samples following a Normal are close to the distribution mean, after the dimensional reduction algorithm, most of those points are placed in one area of the 2-D output layer. In contrary, it can be seen how a reduced number of observations are mapped into the border units, maintaining a Gaussian presence overall. Later on, topology preservation will become a much less remarkable aspect to notice as dimension grows.

To demonstrate this effect with a more solid foundation, let us compute all the Euclidean distances of the unit's codebook vectors (3-D) to the center of our data distribution (zero mean). In the following figure a numeric table shows the euclidean distances where each codebook vector is denoted as "VXX", where "XX" is the unit's number linked to it.

V1	V2	V3	V4	V5
2.2598032	1.9515208	1.5864798	2.1922118	2.6242579
V6	V7	V8	V9	V10
1.8693441	0.9961759	1.3843333	1.7002407	1.9037265
V11	V12	V13	V14	V15
1.7135800	0.8497801	0.7988874	0.9322942	1.8077420
V16	V17	V18	V19	V20
1.9628763	0.9948213	1.2548057	0.6861580	2.0406800
V21	V22	V23	V24	V25
1.8296491	2.0563339	2.1966628	1.9874280	1.4609521

Fig. 3.2. Euclidean distances of codebooks with respect to data distribution mean.

Figure 3.2 shows a table with all the distances computed from each codebook vector and the origin (the mean of the normal distribution). Note how codebooks with numbers 13 and 19 are the closest ones to the origin. This result is expected as graph B from Figure 3.1 shows that these contiguous units hold the greatest number of sample mappings to their input space representatives (codebook vectors).

Previously, *U*-Matrix has been presented as a quality measure (2.6.4), but it may be used for other purposes. In one hand, it can be used as a quality measure to graphically see how the grid size fits the input data. It gives an idea of how the codebooks of each unit are located in the input space. It is known that two pairs of contiguous units may have distant codebooks between the elements of the pair in relation to the other one. If the output layer is saturated with a units excess, several nodes may not have any sample mapped to it. *U*-Matrix tells us if the nodes are distant from each other meaning that,

during training, codebooks have moved to the edges to fit the maximum number of input samples as a result of a lack of input points. In the other hand, *U*-Matrix can be used for clustering or classification purposes. Looking at the *U*-Matrix it is possible to see the limits of the classes contained in the set.

Resuming our Gaussian scenario, *U*-Matrix is shown in graph E from Figure 3.1. In the absence of a classification purpose, it does not make sense to use the *U*-Matrix to uncover the borders of the data set (for this particular example), but it can help see the correct choice of the grid size. Note that the central units are lighter due to its small distance with respect to the surrounding nodes. Majority of input points are mapped into these units. Moving towards the limits of the grid the distance increases. We can say that *U*-Matrix plot and the count map are inversely related as appreciated in such plots. When several observations are mapped to contiguous units in the grid (darker areas in the counts plot), distances among their codebook vectors are small (lighter areas in the *U*-Matrix).

Mapping plot (graph F from Figure 3.1) displays the actual association of each input object to its corresponding unit at the output. Given the actual data set, there is few important information that we can extract from this visualization output. In first place, the distribution of the points within the units does not follow any particular structure. Objects are located at apparently random positions. The only remarkable information (not easy to appreciate) that we can get is the point density of each unit. In such mapping plot it is possible to distinct that the middle node (number 13) presents the highest sample population followed by the surrounding nodes. Border units (e.g. number 1) hold a smaller number of sample counts.

Lastly, the quality plot shown in graph G from Figure 3.1 displays the quantization error of each cell. Recall that such quality measure is computed as the mean distances from all the observations are mapped to a single unit to its codebook vector. Again, unit 13 is filled with a light intensity, meaning that observations mapped to it, are close to its unit's codebook. In the other hand, top left unit is darker, denoting distant input samples with respect to their associated codebook vector. Overall, our grid size choice has been correctly set as the majority of units hold smaller quantization errors.

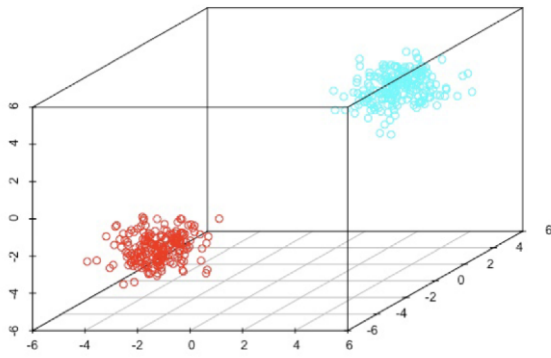
### 3.3.2. Mixture of Gaussians

In the present demonstration a new complexity level will be added. Now, the data set will be formed by two different populations. Particularly, the map will be trained using two 3-dimensional normal distributions centered in an odd symmetric way with respect to the origin of the coordinates system. This demo will allow us to exploit SOM clustering properties, making it a more real-life case.

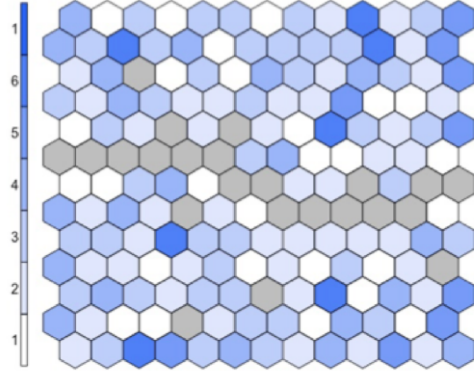
Data has been chosen for the purpose of making the clustering algorithm as optimal as possible. It has been done in such way to increase understanding and to present clear, easy and practical results.

Let us start by describing our input data. Sample set contains  $N = 400$  3-D points. Populations sizes are half of the total set (200 each). First population follows a Normal with mean  $(-3, -3, -3)$  and  $0.5\mathbf{I}_3$  covariance matrix. Second one is centered at  $(3, 3, 3)$  and same covariance matrix as the other one. Data is depicted in graph A from Figure 3.3.

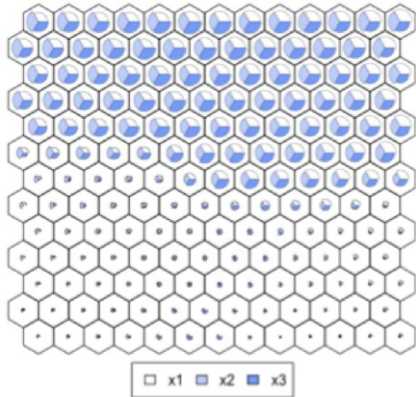
The map will have a grid size of  $13 \times 13$  and units are hexagonal. The choice for the geometrical form of the units does not have a relevant impact in the present example from a results point of view. SOM model will be trained in online mode and with the default number of iterations.



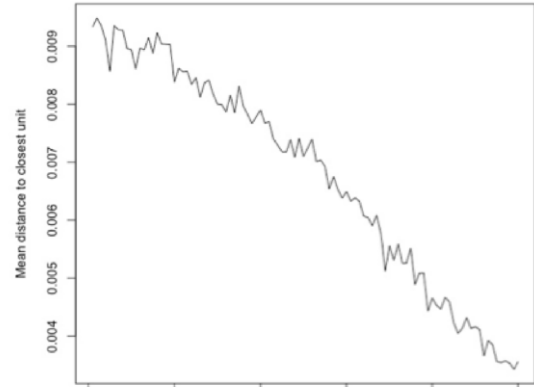
**Original data (A)**



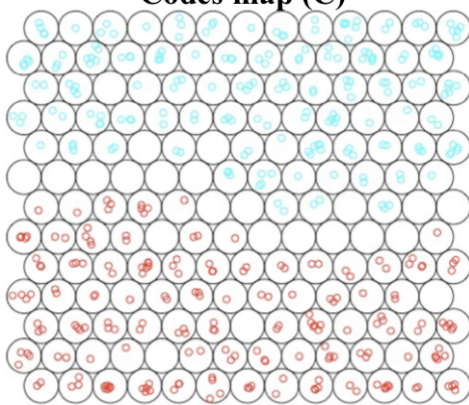
**Count map (B)**



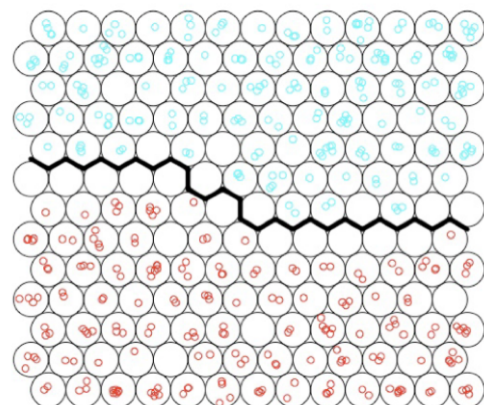
**Codes map (C)**



**Changes plot (D)**



**Mapping plot (E1)**



**Mapping plot (E2)**



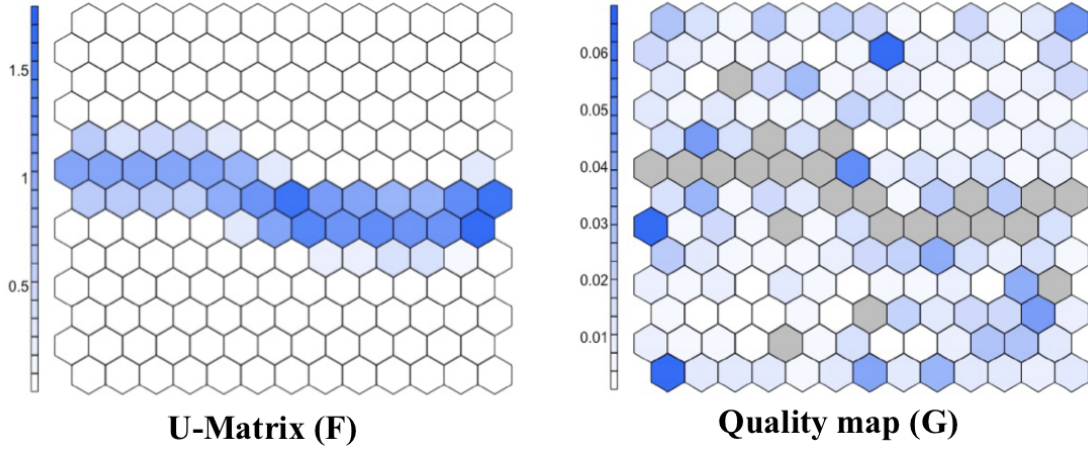


Fig. 3.3. Graph A: Scatter plot of data; graph B: Count map; graph C: Codebook map; graph D: Changes plot; graph E1: Mapping plot; graph E2: Mapping plot with boundary; graph F: *U*-Matrix; graph G: Quality plot.

In graph E1 from Figure 3.3, one can see the mapping plot, in which objects from the sample set are mapped into their corresponding units and colored according to their classes. Note that the class labels are the true labels that have been manually added and not inferred by the algorithm. One can see how SOM, without taking into account data class labels (unsupervised), is able to perfectly separate both populations.

Same separation is manifested in the codes map (graph C from Figure 3.3). Such graphical output, not only let us identify a clear separation from data, it also describes how observation components differ at a local perspective (units) and at a global scope (clusters or classes). Clearly, one can see that the upper cluster belongs to the Normal centered at  $(3, 3, 3)$  by inspecting the codes plot. Sample components from this class hold greater values<sup>10</sup> than observations belonging to the normal with mean at  $(-3, -3, -3)$ , being their sectorial plots of single units noticeably smaller at the bottom side of the grid.

Furthermore it is possible to identify at first glance the free units in which any observation has been mapped to (gray cells depicted in graph B from Figure 3.3). This trail of blank nodes manifest the geometrical separation between both populations. One can anticipate that these nodes will present larger distances to their contiguous nodes. *U*-Matrix plot shown in graph F from Figure 3.3 will reveal the prior fact.

As opposed to the single Gaussian case, changes plot (graph D from Figure 3.3) does not show any horizontal levels manifesting the different phases of the algorithm. Recall that this phases are not formally demonstrated despite the fact of its appearance in some datasets. In the majority of cases these phases appear in datasets from where their samples has been generated from one single population. Being the current example a SOM model applied to two different populations, the map formation phases tend to disappear as, for

<sup>10</sup>We refer to greater values in a non-absolute value way. Positive numbers are greater than negative values.

instance, the ordering phase would be difficult to reveal when codebook vectors try to fit at the same time two cloud points distant to each other. As before, the curve presents a decreasing behavior.

Regarding the quality map depicted in graph G from Figure 3.3, grid size allows a class separation from the input observations. If we were to increase quality under the scope of quantization error, one can reduce the number of cells to, at the same time, decrease the mean distance from all observations to their associated codebook vector.

### 3.3.3. Non linear patterns

Main goal is to prove SOM's performance when applied to non linear structures. Additionally, we will apply a supervised version of SOM, implemented in Kohonen library.

Our choice for the two non linear patterns will be an annulus uniform distribution. Points generated from these distribution are randomly placed in the 2-D space forming a ring shape. Such ring has a circular support with an upper and lower radius limits. Both population will have concentric support functions but with different outer and inner limits.

As in any supervised problem, we will split our data into two different sets: train and test. Train and test sets will have a sample size of  $N_{\text{train}} = 800$  and  $N_{\text{test}} = 200$  observations respectively. Half of the sets will follow two different ring distribution. Figure 3.4 illustrates a scatter plot of both training a test sets.

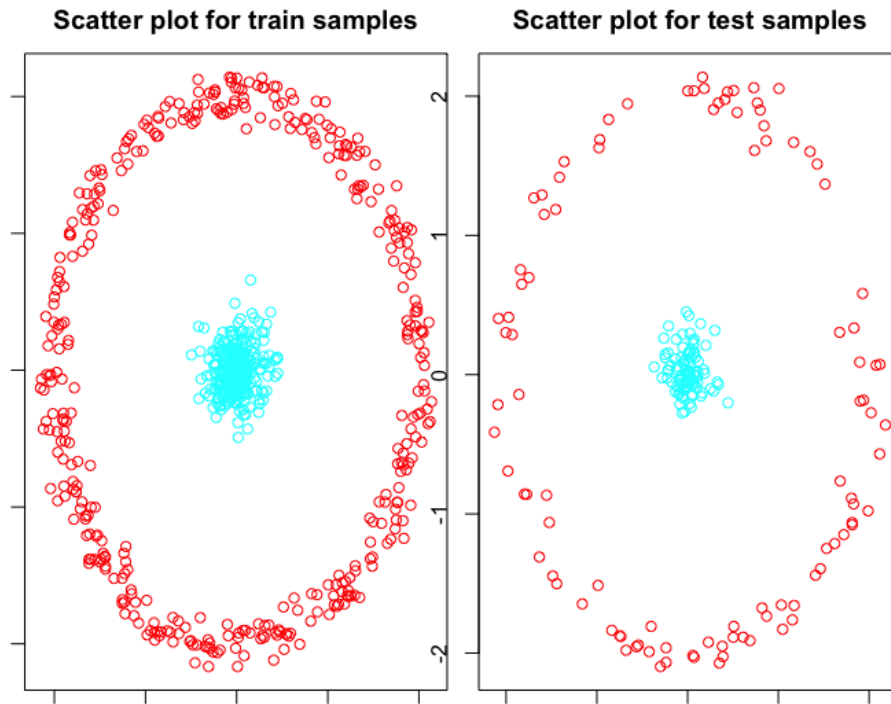
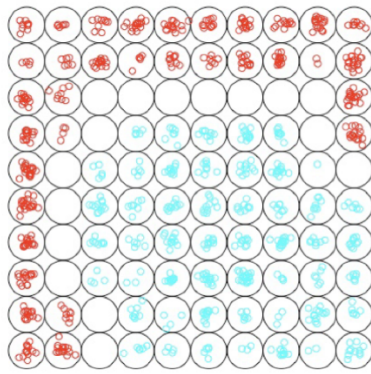


Fig. 3.4. Scatter plots for train and test sets.

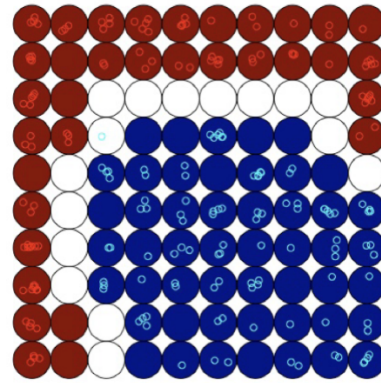
The inner cloud of points is still a ring distribution with the inner radio being practi-

cally zero. Despite this fact, our data structure follows a non linear behavior that other dimensionality reduction techniques find trouble manipulating.

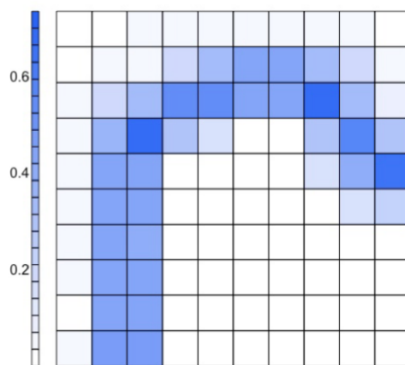
The grid size will have dimensions  $10 \times 10$ . Again, units hold rectangular geometrical form and adaptation (2.3.4) is performed in incremental mode with the default number of iterations.



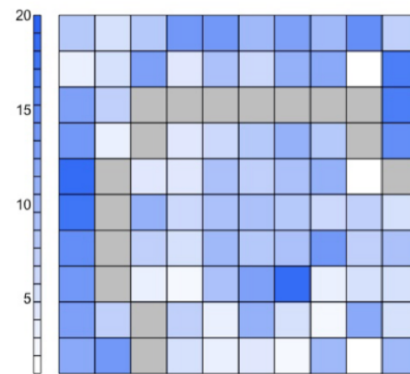
Mapping plot for train set (A1)



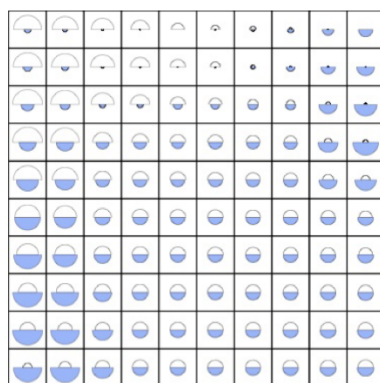
Mapping plot for test set (A2)



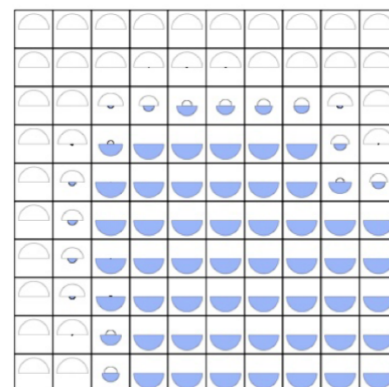
U-Matrix (B)



Counts map (C)



Codes plot for data features (D1)



Codes plot for data labels (D2)



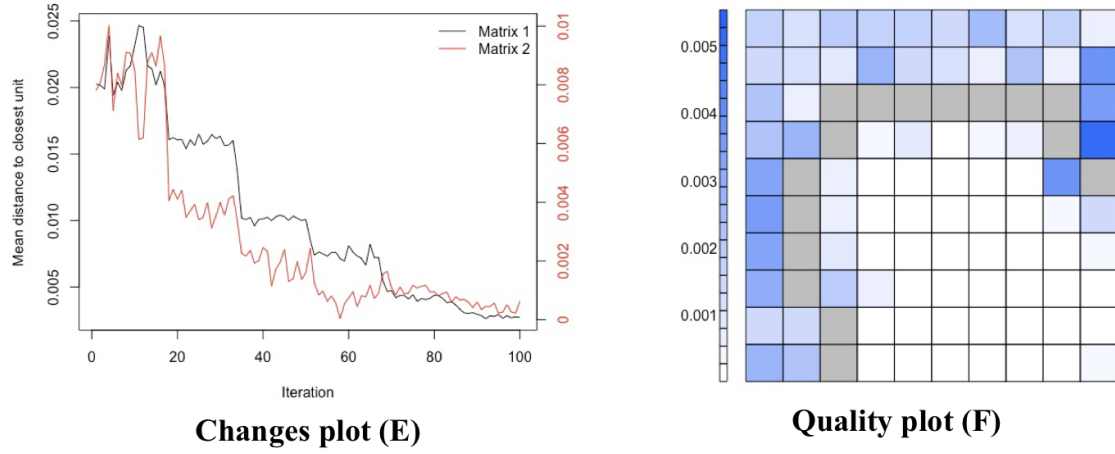


Fig. 3.5. Graph A1: Mapping plot for train set (true labels); graph A2: Mapping plot for test data (true labels and predictions); graph B: *U*-Matrix; graph C: Counts map; graph D1: Codes plot for data features; graph D2: Codes plot for data labels; graph E: Changes plot (two curves for each layer); graph F: Quality plot.

Graph A1 from Figure 3.5 shows the mapped observations to their corresponding units following a color code to distinguish classes. It is easy to recognize that blue dots coincide to the inner ring samples. This manifestation brings us again to topology preservation property, as we see how red dots “embrace” the blue observations in the same way that scatter plot in Figure 3.4.<sup>11</sup>

In general terms we can say that the final output of the algorithm is a grid map conformed by units which, at the same time, are represented by vectors (codebooks) in  $\mathbb{R}^D$  being  $D$  the input data dimension. During learning, these vectors move in the original data space to fit a certain number of training points with the ultimate purpose of describing those observations. Once the mapped is configured, we can map new points to our already trained map and describe them under a compressed form in a new 2-D output space (grid or output lattice).

In graph A2 from Figure 3.5 we can see the mappings of test data fitted under our trained map. Additionally, we can see with the unit’s background color the predictions performed to each observation, also captured in the confusion matrix in the next table:

		Predicted		
		1	2	NA
True labels	1	100	0	0
	2	0	99	1
	NA	0	0	0

<sup>11</sup>We refer to topology preservation as the fact that the data points projected into the lower-dimension space (output grid), maintain their original “topological information”. In other words, if two points are located near to each other in the original data space, same will occur when mapped to the output space. Note that, in the present case, there is not a dimensionality reduction since both spaces are 2-D.

Samples have been perfectly classified with the exception of one blue sample that could not be associated to any category (NA in the table above). Such sample could not be classified since it has been mapped to a unit that serves as a boundary unit to separate both rings. Graphically we can see such limit by looking at *U*-Matrix depicted in graph B from Figure 3.5, where several darker units draw a path with a high distance value to contiguous nodes. In an analogous way, the counts map (graph C from Figure 3.5) reveals the same separation by means of units containing zero observation or counts.

Graphs D1 and D2 from Figure 3.5 display the codes plot for data features and labels respectively. Just like in former examples, this visualization output gives information about how the observation components associated to the units are distributed. Looking at graph D1 it is possible to appreciate the outer and inner rings by distinguishing the variation in the components. For instance, inner ring units are equally distributed regarding their components ( $x_1$  and  $x_2$ ), as opposed to the outer ring units where the  $x_1$  feature presents an inverse variation with respect to  $x_2$ . Graph D2 can be seen as the mapping plot depicted in graph A1, with additional information about the boundary units and the presence of each class in different parts of it.

Note that all this outputs, with the exception of graph A2, have been computed solely taking into account training data. Graph A2 is just a mapping of new data into a learned map.

Graph E from Figure 3.5 presents the changes plot. As said before, supervised SOM computes two different layers: one is associated to the data features and the other uses the data labels. The black curve represents the learning process (measured in mean distance to closest unit) of the feature layer and the red one is that associated to the labels layer. Label layer decreases faster as the information that holds contributes meaningfully more than just the data features. This is mainly the reason why supervised SOM's performance is better with respect the unsupervised version.

Lastly, graph F from Figure 3.5 displays the quality plot. A curious insight that can be extracted is that the inner ring units compress the information better than the outer one. Data points are more distant from their codebooks in the outer ring (darker blue) than in the inner cloud (white).

## 4. CASE STUDY

In this chapter we will apply a SOM to a “real life” dataset. This database has been chosen in order to bring sensible results when applied with SOM. Specifically, the database under study contains multiple features of 18.147 soccer players. This dataset has been used for multiple applications: from clustering (e.g. team management) to estimation (e.g. player salary or market prize) by means of multiple techniques. The use of SOM applied to such database can reveal insights and new perspectives that other approaches would find difficult to extract and, above all, challenging to visualize and interpret. Exploiting SOM’s dimensionality reduction capability and taking advantage of its visualization outputs, results can be presented in a concise and explainable manner.

The main goal that will be pursued regarding the study of this current database is the classification of players with respect to their field positions. For that aim, a feature selection process, taking advantage of several visualization outputs, will be carried out.

### 4.1. FIFA 19 complete player dataset

Every year the gaming company ELECTRONIC ARTS SPORTS develops a new version of the popular multi-platform game FIFA. With each release, the game has achieved a more realistic experience, making it as consistent as possible with the real world soccer. This, not only applies to the general game but also to the actual players. Such detailed player characteristic manifested in the game is been possible due to an exhaustive labeling process done by experts in the field. The procedure yields a fully descriptive database that can be used for statistical examination.

The original data set, scraped from <https://sofifa.com/> using Shrivastava (2017)<sup>12</sup> is composed by  $N = 18.147$  observations, each of it holding 82 attributes. Many of these features are not useful for our study goal, therefore they have been omitted. Player soccer skills have been manually rated (by soccer experts) in a scale from 0 to 100. Goalkeepers have been omitted in this analysis.

A data formatting process has been carried out, where several aspects has been treated. Firstly and regarding our goal (players classification according to their field positions), a new variable has been created. An extra column has been added to specify the simplified position category. Such modification is convenient as there exists 26 player’s position variants (8 forwards, 11 midfielders and 7 defenders). Analysis based on this new variable is simplified and more generic.

Regarding the management of NAs, two filling strategies have been made. Missing categorical attributes have been replaced by the mode and numerical values by its median.

---

<sup>12</sup>Retrieved from <https://www.kaggle.com/karangadiya/fifa19>.

Median has been chosen to avoid noise caused by outliers<sup>13</sup> present in the database.

## 4.2. Classification of soccer players with respect to their field positions: part 1

Intuitively, as the title explains, the aim is to classify players under three different classes corresponding to their field roles: forward, midfielders and defenders.

Let us first take advantage of the visualization outputs to understand our data. It will be done by showing the properties map, explained in Section 3.2, which was not computed in the former demos due to its irrelevancy, as data features used in those examples did not have any particular meaning. For this purpose, a map will be trained using a small data sample of  $N = 1.000$  players. A supervised SOM model has been used to improve performance.<sup>14</sup>

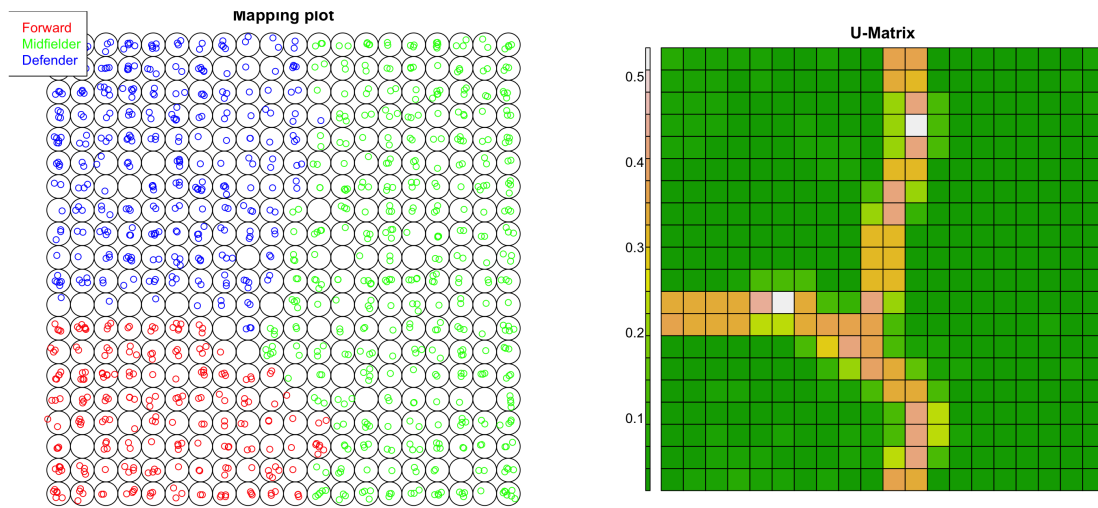


Fig. 4.1. Left graph: Mapping plot of 1.000 soccer players with their corresponding (true) class in color code. Right graph: U-Matrix plot.

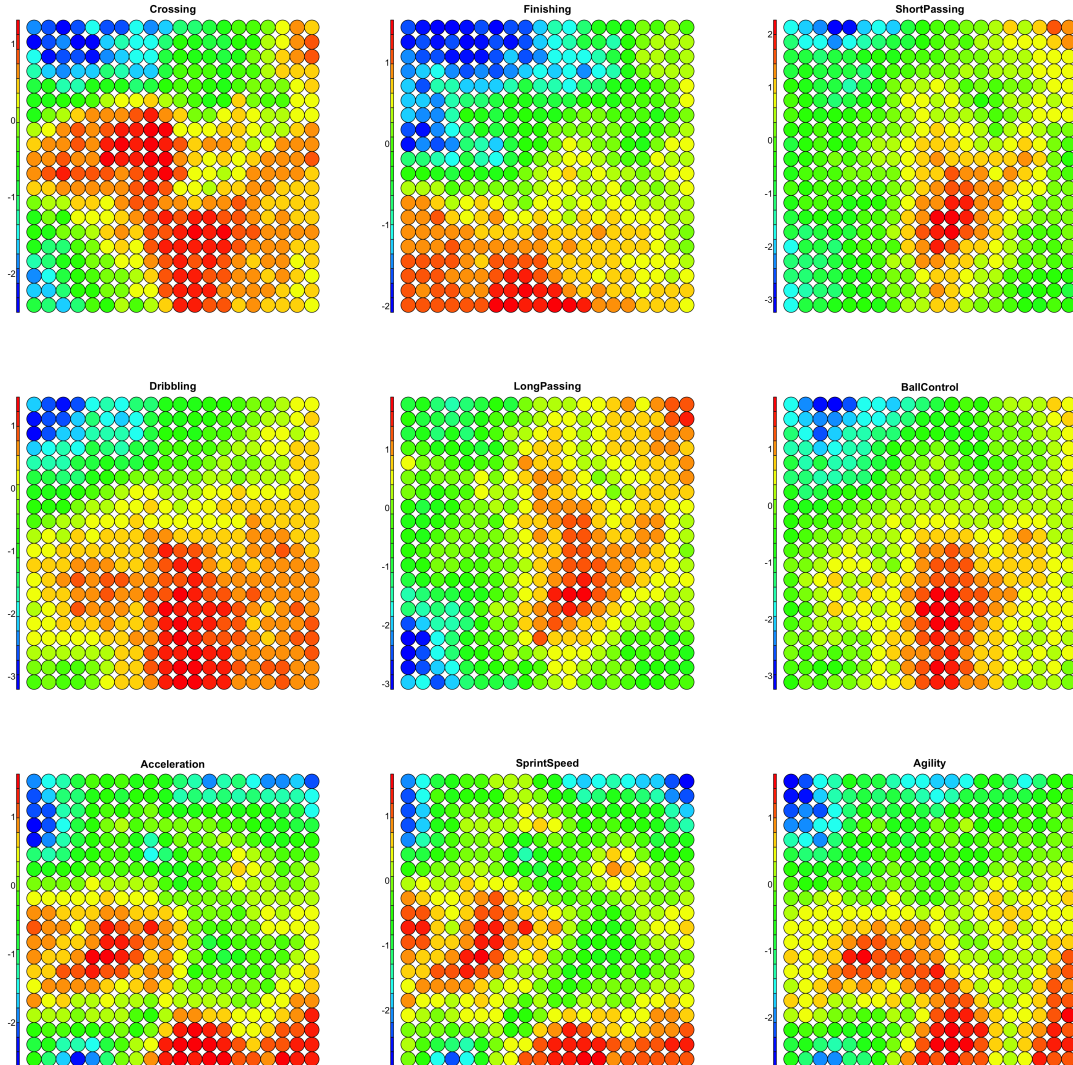
Properties map consists in a heat-map that allows the user to inspect the map under one property or data feature. In order to avoid a saturated figure with all the player skills, this map has been trained using a set of 19 variables, sufficient enough to accomplish a decent separation among classes. The model is generated upon a rectangular grid with dimensions  $20 \times 20$  and using a supervised SOM version.

The left graph from Figure 4.1 depicts the mapping plot, showing the true class of each observation in color code. Note that the gap between defenders (in green) and forwards (in blue) is more prominent than between the lasts and midfielders. In order to inspect

<sup>13</sup>Observations that significantly differ from the rest and add noise to the model.

<sup>14</sup>If an unsupervised SOM was trained, class separation would not be possible just by taking into account the player skills. Some of the midfielder's variations hold similar profiles as forwards or defenders. Adding a second layer with labels, drastically improves the results.

and understand the properties map, is necessary to see the actual mapping of the classes in the grid as the mapping plot shows. Moreover, the  $U$ -Matrix plot depicted in the right graph of Figure 4.1, manifest the same separation boundary seen in the left plot.



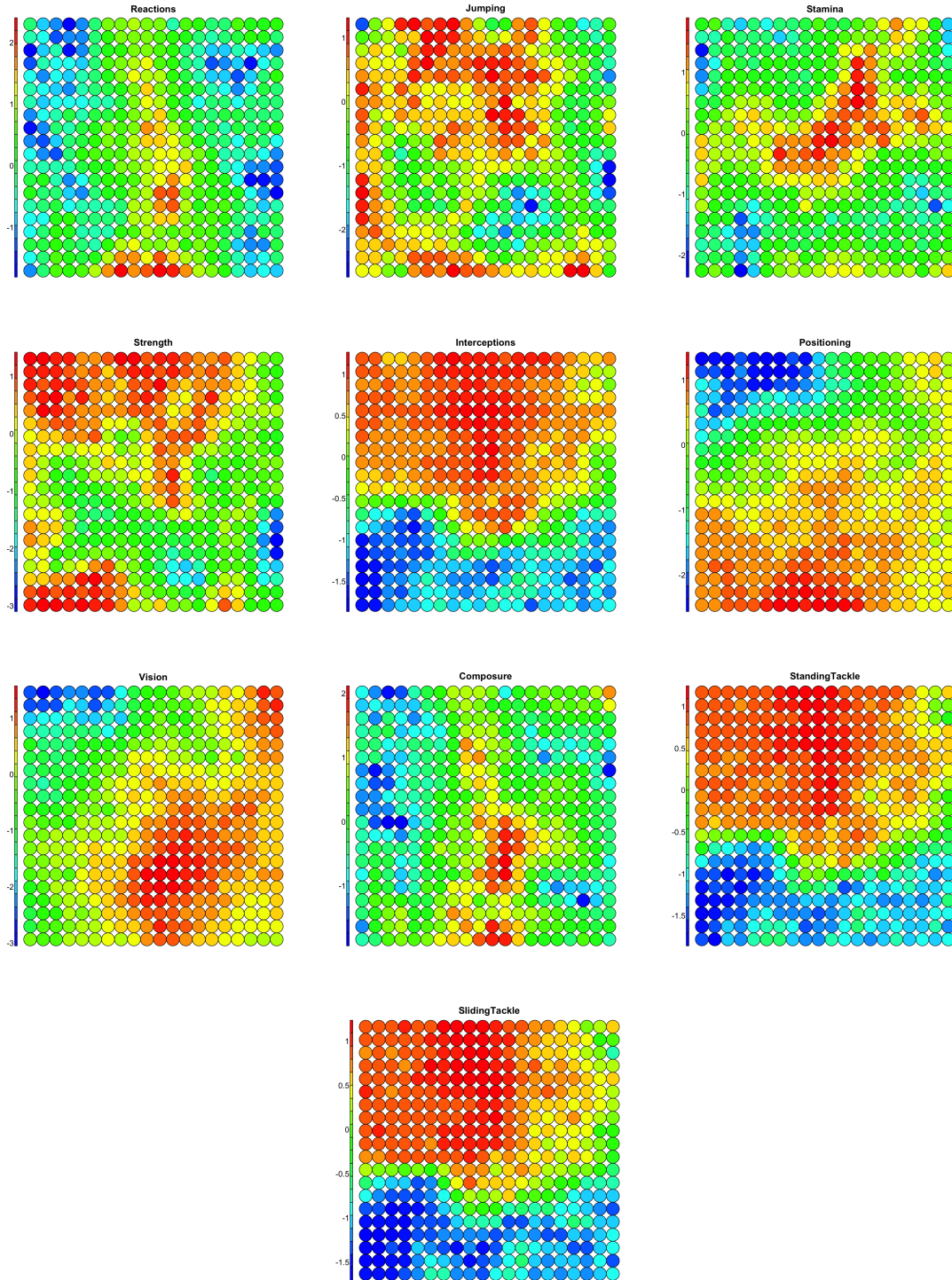


Fig. 4.2. Properties map of 19 features representing soccer skills. Blue areas denote a small presence of that attribute among the observation mapped to that areas. As the color gets close to a red tone, the presence of that variable increases.

The figure above presents 19 property maps, each of them representing one attribute along with its presence over the observations mapped to the output grid shown in the mapping plot. This visualization map is SOM's biggest edge compared to other dimension-



reduction techniques. At first glance one can see the presence of the complete attributes list distributed in the trained map. It can be thought as a codes plot that has been disaggregated and presented in a heat-map form to inspect each variable independently. The two main applications for this visualization map is feature selection or general understanding of the dataset.

Several insights can be extracted from the properties plot; let us describe a few of them. The first thing than one can notice is the bigger presence (red areas) of offensive skills located in the areas where forwards were mapped in the grid (bottom left corner). For instance, the *finishing* and *dribbling* property maps (Figure 4.2) present a stronger appearance in the forward area (bottom left corner). Same occurs with the defensive and midfielder's attributes. Moreover, those skills lose their presence as we move further from this area, manifesting which observations, within the same cluster, hold an offensive or defensive style. In other words, it is possible to observe a second level clustering that reveals the variations of the three main field positions. Furthermore, property maps can reveal interesting conclusions that would be hard to evaluate with other approaches. This is the case of *reactions* property map (Figure 4.2). Such map shows separated low presence areas (blue clouds) even in the same cluster, denoting a lower relevance of that attribute in those player profiles and revealing sub-clusters that most likely are associated to position variations of that class.

Let us resume our initial goal for this section: classification. Properties map can be used to this purpose. So as to obtain small prediction error it is important to select a set of attributes that do not introduce any negative effect to our model. If we trained the model using the whole set of features we would be introducing redundant and irrelevant information to build it. One way of proceeding regarding feature selection is by inspecting property maps. The goal is to choose features that present similar variations in the map according to the class separation observed in the mapping plot in Figure 4.1. For instance, if we take a look at *stamina* property map (Figure 4.2), its variance regarding its presence does not add relevant knowledge to the model in order to classify the player's positions. In the other hand, *sliding tackle* property map (Figure 4.2), almost imitates the class separation manifested in the mapping plot depicted in Figure 4.1. Thus, adds valuable information to the map, enhancing predictions. Same occurs with *standing tackle* property map (Figure 4.2) although if it is included as a variable for the model to take into account, it would be redundant being practically identical to *sliding tackle* property map.

Taking into account the previous paragraph, a new set of 6 attributes have been selected to feed our classification model. These variables are: *finishing*, *dribbling*, *ball control*, *interceptions*, *positioning* and *standing tackle*. Two sets have been created: training set with  $N_{\text{train}} = 1.000$  and test set with  $N_{\text{test}} = 415$ .

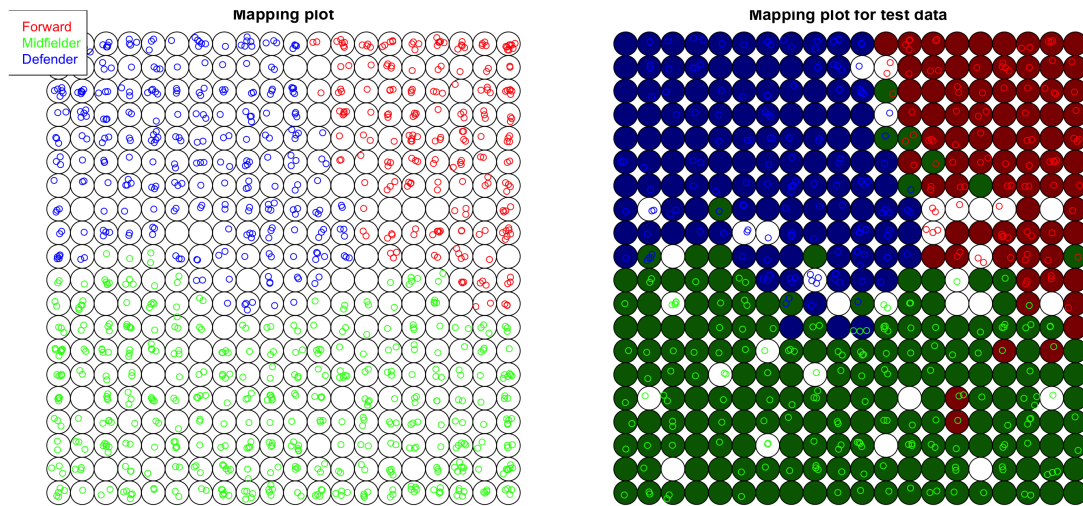


Fig. 4.3. Left graph: Mapping plot of 1.000 soccer players (train set) with their corresponding (true) class in color code (second model). Right graph: Mapping plot of 415 soccer players corresponding to the test set. Predictions are represented by the unit's background color and true labels are so by means of the observation colors. White units correspond to non-categorized nodes.

In Figure 4.3 (left graph) above, the mapping plot of the training set is shown. The separation between classes has been as precise as in the former case (Figure 4.1) but with different cluster localization within the map. Next to it appears the mapping plot of the test set along with its predictions. The confusion matrix associated to such predictions can be seen in the table below.

		Predicted			
		Forwards	Midfielders	Defenders	NA
True labels	Forwards	91	3	0	8
	Midfielders	6	150	3	11
	Defenders	0	5	133	5
	NA	0	0	0	0

Overall, the model has been able to perform a proper separation with the exception of some points that were wrongly classified or labeled as a null category (NA). Regarding the samples categorized in a wrong class, it was an expected outcome due to the simplification of soccer player positions in just three categories. Non-labeled samples could not be predicted since they fell in an unit which did not have any observation associated to it at the end of the learning phase. Note that the white units at the right hand side of Figure 4.3 are the same units with no samples mapped to it in the mapping plot located at the left in the same figure.

One approach that could be done to improve predictions is to refine the train set in



order to maximize the statistical description of the whole set. Performing a random partitioning of the set (as done in this example) can bring a risk of having a high degree of redundancy in the train set. Instead of picking samples randomly and with any statistical criteria, ensure a maximum variation regarding components within the train set in order to avoid blank units in the trained map as occurred in Figure 4.3. SOM's performance decreases much more in relation with other techniques when a considerable amount of redundant data is used to build the model.

### 4.3. Classification of soccer players with respect to their field positions: part 2

In Section 4.2 the property maps revealed the existence of second level clusters within the same first level class. By means of inspecting each attribute in the property maps, one could notice, specially regarding midfielders, the existence of sub-clusters that revealed the degree of offensive or defensive styles of the players. This is coherent with the prior knowledge before constructing the map, since the total set of players could not be only classified in three classes. There existed multiple variations within the same player role in the field.

In other unsupervised scenarios, a data analyst could be applying a certain clustering model and notice that the initial number of clusters, chosen to model the data, could not be enough for a proper grouping. In such case, the correct proceeding would be increasing the number of clusters to enhance the degree of description of the model. This is precisely the aim of this section, with the exception that we had a prior knowledge of the data (supervised).

A SOM model will be trained for the same purpose as in Section 4.2 (i.e. classification), however, the set of classes will be expanded as the result of inspecting property maps, manifesting second level classes. The categories in this model will be: forward, attacking midfielder, neutral midfielder, defensive midfielder and defender. Being the current a supervised model, the new classes are manually introduced in the data set using the actual player positions.

The map will have a rectangular grid of size  $20 \times 20$ . In order to capture the subtle characteristics between the new profiles, the train set size has been increased to  $N_{\text{train}} = 3.000$  observations. Sample size for the test set is now  $N_{\text{test}} = 1.314$  players. The initial 19 features used in the first approach of Section 4.2, has been taken into account to build the current model.

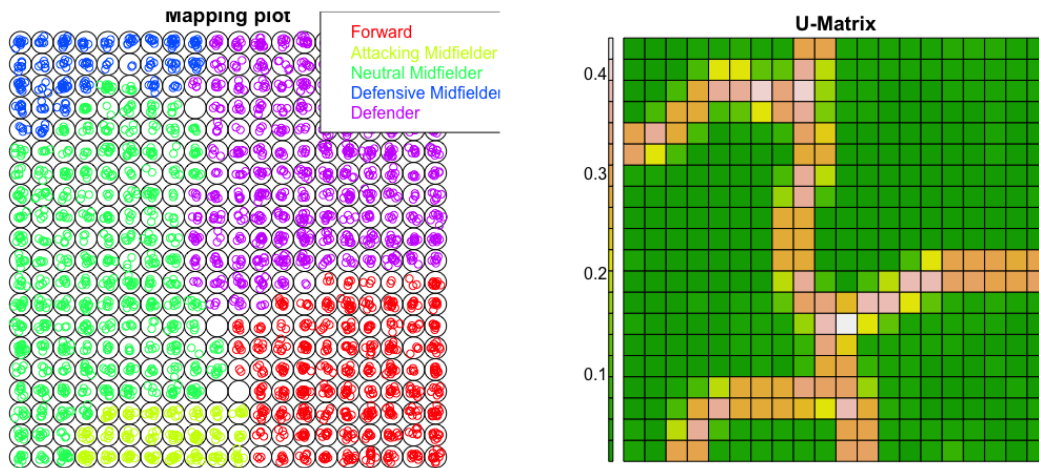


Fig. 4.4. Left graph: Mapping plot of 3,000 soccer players with their corresponding (true) class in color code. Right graph: *U*-Matrix plot.

Above, Figure 4.4 shows both the mapping and the *U*-Matrix plot of the training set. Again, supervised SOM allows a class separation. Looking at the *U*-Matrix plot, one can see the limits of each class. When SOM is used in classification, it is important to reduce the thickness of the borders showed in the *U*-Matrix plot. The more units that constitute such borders (light brown units in right graph in Figure 4.4), the less information the model will have to predict an new observation when mapped to border units. In order to improve predictions, the grid size could be shrank so that the “vacuum” between clusters is filled with more information and therefore, reducing the number of border units. Additionally, more points could be fed to the model in order to fill the information void of the class borders.

		Predicted				
		Forward	A. Midfielder	N. Midfielder	D. Midfielder	Defender
True labels	Forward	202	15	43	0	0
	A. Midfielder	22	20	47	2	1
	N. Midfielder	63	35	237	46	23
	D. Midfielder	0	1	61	23	19
	Defender	0	0	30	15	410

The table above shows the confusion matrix of the train set mapped to the trained map. The mapping plot can be seen in Figure 4.5. As opposed to the former case, presented in Section 4.2, every observation has been assigned to a class (right or wrong), therefore, no samples have been mapped to an empty unit. Predictions have been acceptable (yet improvable) for defenders and forwards. Regarding the midfielders divisions, the predictions were not as solid as expected. Two main reasons could explain such poor behavior. Firstly, the train set used was not equally distributed in relation to classes. This can be seen looking at the total number of observations per class shown in the confusion

matrix. SOM was not able to statistically compress the different midfielders roles as a consequence of insufficient data points from those classes. Secondly, *as an amplifier*, the positions showed in the original data base could not be accurate enough. Some players can be categorized in more than one field position and, in the present data set, only one label can be assigned to each player regarding its position. It can be seen as a human bias that is inherited in the model.

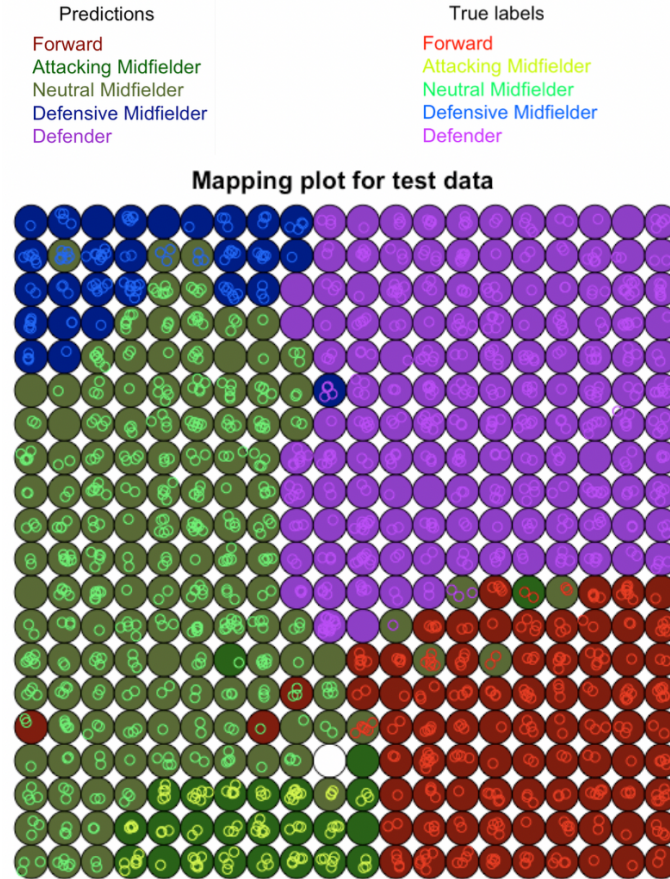


Fig. 4.5. Mapping plot of the test set with predictions.

To improve results, further tasks could be done. As a first approximation is important to feed the model with equally distributed data so that any class becomes underfitted into the model. Secondly, tuning the grid size could also improve results. An useful measure that can be used to address the selection of the grid size is the *U*-Matrix. Lastly, if the prior changes did not improved the performance, a feature selection process could be done using property maps as showed in Section 4.2.

#### 4.4. Case study conclusion

By applying SOM to this particular database, it has been possible to inspect a high-dimensional space by transforming it into a visualization grid, much more manageable and interpretable. Such reduction has been performed keeping (utmost) the topological

information between the samples at the input space, as an advantage over, for instance, PCA<sup>15</sup>.

The supported visualization outputs than SOM computes, allow the extraction of patterns, clusters and associations among data which is organized within non linear structures. Such elements can be converted into actionable insights to be applied for different goals. An example of this is the utilization of property maps to extract design-related parameters to improve the model. With the aid of property maps we have been able to select a subset of features that improve the classification. Additionally, new classes have been inferred to create a more descriptive grouping regarding the players.

The current case study shows a simple but effective guidance structure that can be reused in other application.

---

<sup>15</sup>PCA creates a space by the generation of new axis build upon the orthogonal maximum variability directions of the original data. When points are projected to the new space, topology information from the original space is lost.

## 5. CONCLUSIONS

This Bachelor Thesis gives a global in-depth description of Self-Organizing Maps, and numerous practical demonstrations to support the theoretical content given. It can be used as a pedagogic manual for starters whose desire is to understand the technique from a global perspective, avoiding saturated mathematical explanations, and complement the theoretical foundations with simple *ready-to-use* examples that enrich the learning process. The coded examples are accessible from the GitHub repository provided at the beginning of this document.

The mathematical formulation provides a solid backbone from where the contents can be further expanded. The aim was to simplify the complex ideas behind the original formulation of the technique by means of detailed explanations. Such simplification may have brought other negative side effects such as superficial descriptions regarding some complex mathematical scopes. Further improvement on this aspect could be done by giving more graphical content, supporting the mathematical descriptions and avoiding dense explanations.

On the subject of graphical and numerical illustrations, the main aim was to portray the practical behavior of SOMs under two different scopes: statistical (using synthetic data) and real life applications (by means of an interpretable dataset). Concerning the first scope, three demos were presented in order to highlight and present different aspects of SOMs. The simple Gaussian case gives a practical view on the visualization outputs, giving statistical meaning to each of them, manifesting the studied properties addressed in the theoretical foundations. The mixture of Gaussians case adds a new complexity level that allows the exploitation of other visualization properties. Non linear patterns shows the performance of SOMs when applied to certain data that limits other dimensionality reduction approaches such as PCA. Adding new demos with different data distributions could be an improvement in this section.

On the other hand, the case study allows a “real life” procedure applying SOM to a familiar dataset. The reader can observe the interpretable capabilities of a SOM’s model with high dimensional data. Obtaining optimal results was not the objective during the case study. The main goal was to present insights on how to use the output maps to select features for an optimal classification and extract deep (previously known) patterns hidden within the data in form of clusters. Further exploration using the same dataset could be done to amplify the possibilities of SOMs.

A general aspect to be improved is the inclusion of harder examples in order to give a detailed description of limitations and the effect of the several tuning parameters, such as the grid size and the topology of the grid. Additionally a comparison between other dimensional-reduction techniques, such PCA or MDS, could be done.

In a nutshell, SOMs can be used to boost interpretability over complex structures as a consequence of a transformation to a *human-friendly* output, created upon a high dimensional space. Such capability can be used mainly in unsupervised clustering problems to extract insights by means of focusing on preserving the topological similitude of the original data.

## BIBLIOGRAPHY

- Akinduko, A. A., Mirkes, E. M., and Gorban, A. N. (2016). SOM: Stochastic initialization versus principal components. *Information Sciences*, 364-365:213–221.
- Angeli, S., Quesney, A., and Gross, L. (2012). Image simplification using kohonen maps: Application to satellite data for cloud detection and land cover mapping. In Johnsson, M., editor, *Applications of Self-Organizing Maps*. InTech.
- Bauer, H. and Villmann, T. (1997). Growing a hypercubical output space in a self-organizing feature map. *IEEE Transactions on Neural Networks*, 8(2):218–226.
- Bellman, R. (2015). *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press.
- Benítez-Pérez, H. and Benítez-Pérez, A. (2010). The use of wavelets feature extraction and self organizing maps for fault diagnosis. *International Journal of Innovative Computing, Information and Control*, 6:13.
- Disposición 542 del BOE núm. 15 (2017) (2017). Agencia Estatal Boletín Oficial del Estado. <https://www.boe.es/boe/dias/2017/01/18/pdfs/BOE-A-2017-542.pdf>.
- Hastie, T., Friedman, J., and Tibshirani, R. (2001). *The Elements of Statistical Learning*. Springer New York.
- Hebb, D. (1949). *The Organization of Behavior: A Neuropsychological Theory*. A Wiley book in clinical psychology. Wiley.
- Helsinki University, T. (1992). World poverty map. <http://www.cis.hut.fi/research/som-research/worldmap.html>.
- Huliane, C. and Flavius (2012). A Self-Organizing map based strategy for heterogeneous teaming. In Johnsson, M., editor, *Applications of Self-Organizing Maps*. InTech.
- Intellectual Property no 28 (1995). Agencia Estatal Boletín Oficial del Estado. <https://www.boe.es/buscar/act.php?id=BOE-A-1996-8930#top>.
- Kohonen, T., Kaski, S., Lagus, K., Salojärvi, J., Honkela, J., Paatero, V., and Saarela, A. (2000). Self organization of a massive document collection. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 11:574–85.
- Kohonen, T., Schroeder, M. R., and Huang, T. S., editors (2001). *Self-Organizing Maps*. Springer-Verlag, Berlin, Heidelberg, 3rd edition.

- LaTeX (1984). LaTeX a document preparation system. <https://www.latex-project.org/>.
- Martínez-Martínez, J. M., Escandell-Montero, P., Soria-Olivas, E., Martín-Guerrero, J. D., and Serrano-López, A. J. (2016). A new visualization tool for data mining techniques. *Progress in Artificial Intelligence*, 5(2):137–154.
- Oy, L., Himberg, J. V. J., Himberg, J., Alhoniemi, E., and Parhankangas, J. (2000). SOM Toolbox for MATLAB 5. <http://www.cis.hut.fi/somtoolbox/>.
- Pözlbauer, G. (2004). Survey and comparison of quality measures for self-organizing maps. *Proceedings of the Fifth Workshop on Data Analysis (WDA04)*, pages 67–82.
- R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
- Resta, M. (2012). Graph mining based SOM: A tool to analyze economic stability. In Johnsson, M., editor, *Applications of Self-Organizing Maps*. InTech.
- Rojas, I., Joya, G., and Catala, A. (2015). *Advances in Computational Intelligence: 13th International Work-Conference on Artificial Neural Networks, IWANN 2015, Palma de Mallorca, Spain, June 10-12, 2015. Proceedings*. Number parte 2 in Lecture Notes in Computer Science. Springer International Publishing.
- RStudio Team (2019). RStudio: Integrated Development Environment for R. <https://www.rstudio.com>.
- Saevanee, H., Clarke, N., and Furnell, S. (2012). Multi-modal behavioural biometric authentication for mobile devices. In *Applications of Self-Organizing Maps*, volume 376. InTech.
- Shrivastava, A. (2017). fifa18-all-player-statistics. <https://github.com/amanthedorkknight/fifa18-all-player-statistics>.
- TexStudio (2019). TexStudio integrated writing environment for creating latex documents. <https://www.texstudio.org>.
- Vahid Moosavi, Sebastian Packmann, I. V. (2014). A python library for self organizing map (som). <https://github.com/sevamoo/SOMPY>.
- Venna, J. and Kaski, S. (2001). Neighborhood preservation in nonlinear projection methods: An experimental study. In *Artificial Neural Networks — ICANN 2001*, volume 2130, pages 485–491. Springer Berlin Heidelberg.
- Wehrens, R. (2011). *Chemometrics with R: Multivariate Data Analysis in the Natural Sciences and Life Sciences*. Use R! Springer Berlin Heidelberg.



- Wehrens, R. and Buydens, L. M. C. (2007). Self- and super-organizing maps in R: The kohonen package. *Journal of Statistical Software*, 21(5):1–19.
- Zhang, J. and Fang, H. (2012). Using self-organizing maps to visualize, filter and cluster multidimensional bio-omics data. In Johnsson, M., editor, *Applications of Self-Organizing Maps*. InTech.